MYVIEW: CUSTOMIZABLE AUTOMATIC VISUAL SPACE MANAGEMENT FOR MULTI-STREAM ENVIRONMENT

BY

BIN YU

B.S., Tsinghua University, China, 2000

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

ABSTRACT

Recent years witnessed the emergence of more “multi-stream” Multimedia applications, such as multi-site distributed video conferencing, multi-perspective IPTV, and virtual touring of 3D digital worlds from multiple view points, etc. This trend is the result of greater availability of high performance computing devices, high speed Internet access and high quality video/audio capture devices.

In this work, we study the problem of ***visual space management*** in such applications. The key observation is that there are multiple *visual streams* that need to be presented to the users, while the visual space is limited both by availability of physical display devices and by user’s visual attention space. We argue that the visual space is a type of critical resource that requires careful management in similar way as other resources (such as CPU cycles and memory slots). Specifically, the unique challenges are how to model the visual streams and the viewers’ preference among them, how to determine which of these visual streams should be presented to the viewer during the runtime of an application session, and how the output windows of these selected streams should be arranged on the limited display devices.

Many existing systems have explored the solution space from different perspectives. Some systems simply squeeze all available visual streams onto the display device; some systems let the viewers manually manage the screen layout for what they want to watch; some systems resort to professional human editors for help; some systems employ intelligent decision engines to automate the stream selection process by computers. Overall, none of the existing systems

provide a solution that is *customized* to user interest, *automatic* to avoid unnecessary user input, and yields *high visual quality screen output*.

In this dissertation, we develop a 3-phase visual space management framework called *MyView* that makes the following three key contributions. First, a generic shot-based visual stream model and a score-based viewer interest model are designed to represent user interest, which lays the foundation for automatic visual space management decisions. Second, an automatic stream selection algorithm is developed that maintains a timed automaton to calculate which visual streams should be selected for display. As opposed to manually specified automaton used in related work, our approach automatically translates the viewer interest into a timed automaton and so adapts to changes in viewer interest dynamically during the runtime. Third, a screen layout calculation algorithm is designed to compute the screen layout for presenting the selected visual stream(s) on one or multiple display devices in a visually pleasing way that maximizes screen space utilization.

We have built a lecture recording and broadcasting system on top of the ConferenceXP platform to evaluate our solution. Results from computer simulation experiments and a mid-scale user study have confirmed the functional correctness and usability of the MyView framework.

To My Parents Haiting Yu and Yurong Tang

To My Wife Pei Gao

Acknowledgments

First and foremost, my deepest gratefulness goes to my thesis advisor Professor Klara Nahrstedt for her invaluable guidance and support throughout my Ph.D. study. Her insights and suggestions have enlightened me not only on academic thinking and technical problem solving, but also on how to lead a meaningful and productive life by hard work and a high standard on oneself. To me, she has been a great advisor and an irreplaceable role model. I will always be grateful for her consistent encouragement and belief in me during my periods of doubt and frustration. I feel extremely lucky to have Klara as my advisor, who is always dedicated to the success of her students.

I would like to thank the members of my thesis committee: Professor Roy Campbell, Professor Thomas S. Huang, and Professor Brian P. Bailey, for their helpful discussions on my work. Their insightful comments on my thesis have greatly helped me to improve the quality of this work. I also want to thank all the members of Microsoft Research who gave me great help and support during my internships. Especially, I really appreciate the guidance and advice given by my mentors Wei-Ying Ma and Hong-Jiang Zhang from Microsoft Research Asia; my mentor and mangers Yong Rui and Cha Zhang from the Communication and Collaboration Systems (CCS) Group in Microsoft Research Redmond; my mentors and managers Patrick Bristow, Jason Van Eaton and Chris Moffatt from the ConferenceXP Team of Microsoft Research. I would also like to extend my special thanks to Professor Yuan Xue and Professor Yi Cui from Vanderbilt

University and Professor Dongyan Xu from Purdue University for their invaluable advice on how to conduct valuable research. Working with them has greatly benefited my graduate study and research career.

I owe my work to the several staff members from University of Illinois who kindly helped me with my research. Especially, Anda L Ohlsson has been helpful all through my PhD program. Andrew James Macgregor and Scott V Cimarusti helped me a lot in setting up the lecture room environment we used for lecture recording and user study. Vicky L Gress and Sue Rhodes helped me with the logistics of application of user study.

I am grateful for the help and support from all current and former members of the MONET (Multimedia Operating Systems and Networking) research group. Particularly, I would like to thank Wanmin Wu, Muyuan Wang, Hoang Nguyen Viet, Ying Huang, Wenbo He, William Conner and Jin Liang for helping me with the user study and many insightful discussions and fruitful collaborations. My gratitude also goes to Thadpong Pongthawornkamol, Long Wu, Jingwen Jin, Wanghong Yuan, Xiaohui Gu, Duangdao Wichadakul, Jun Wang, King-Shan Lui and Won Jeon.

Last but not least, my parents deserve special recognitions for their unconditional love and continuous support. Without them, all that I have achieved would not have been possible. I am also grateful to my wife Pei for her love and support during my graduate study. Her companionship makes this long journey a joyful experience. This dissertation is dedicated to them.

This work has been funded by the following grants. Without their support, we could not have done such an interesting and challenging task: NSF SCI 05-49242, NSF EIA 99-72884, NSF CNS 05-09314, NSF CNS 05-20182, Microsoft Gift 434U4W – 628267-434028-191100, Carver

Fellowship at University of Illinois.

TABLE OF CONTENTS

Chapter 1. INTRODUCTION

1.1 Motivation

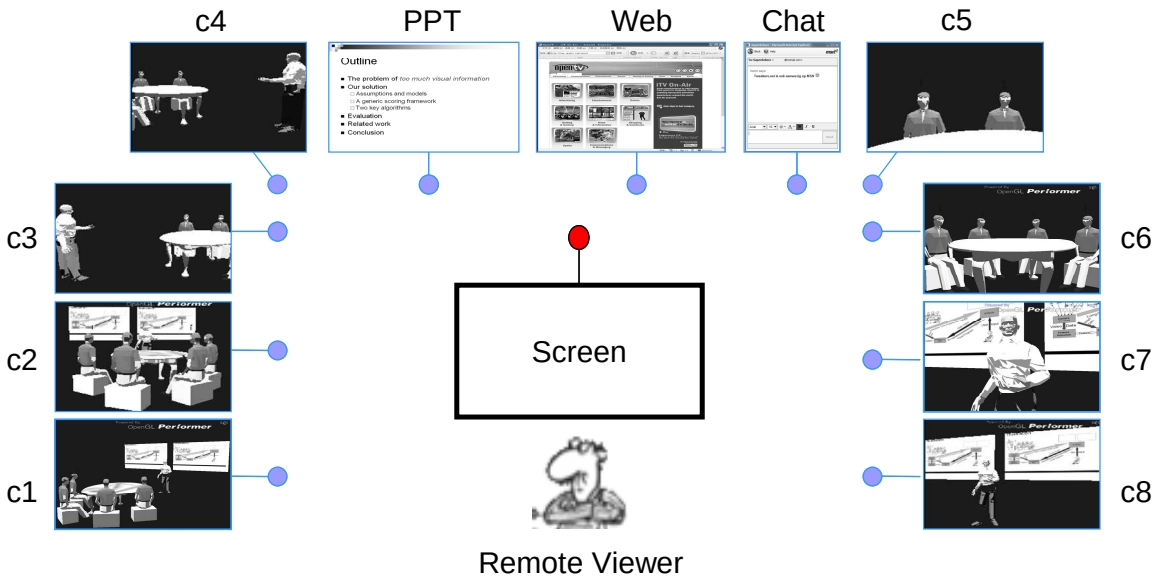

Figure 1. Multiple Visual Streams to Be Rendered On Limited Screen Space

Traditional Multimedia applications normally involve only *one visual stream*: with traditional Video-On-Demand service, one video stream is sent to each viewer; traditional two-way video conferencing systems only render the remote participant in one video window. However, it has become more and more common for Multimedia applications to manage and present *multiple visual streams* for viewers today. For example, as illustrated in Figure 1, in today's distributed conferencing systems, typically multiple visual streams are carrying related visual information to the remote viewers through a limited screen space: the visual streams may consist of live

camera streams showing local and remote participants (*c1* to *c8* in Figure 1), a PowerPoint slideshow, a messenger-like application window for text chat, a web browser for WWW co-browsing, and a whiteboard for sharing quick drawings, etc.. The screen space normally takes the form of one or two display devices. For another example, it has become more and more popular today for a live social event, such as a sports game or art performance, to be captured by tens of video cameras, which form a large set of concurrent visual streams to be presented on each audience's TV screen.

Despite this trend of increasing quantity of visual content provided to users, the *visual space* for presenting these visual streams to a user remains limited for the following two reasons. First, the available *screen space* remains limited. Normally a user would only use one or two display device with an average resolution to view the visual information, such as PC monitors, plasma displays, projector screens or TV monitors. More importantly, a user's *visual attention space* remains to be limited. No matter how many visual streams are presented on one or multiple display devices, a user's visual focus would only be confined to one visual stream window or even a portion of it at any time point.

Therefore, in any application that presents multiple visual streams to a user, there is the problem of *visual space management*: how to allocate the limited visual space to present these visual streams. We envision that this will become one of the most important question in the Multimedia field in the future because more and more Multimedia applications will involve the visual space management problem, especially as content acquisition and distribution become easier and cheaper.

There is no easy answer to this question, because the success of any solution will depend on many factors. We have identified some of the key factors that must be considered in the evaluation of any solution for visual space management, as listed below:

- **User Interest**: since there are multiple complementary visual streams to be rendered in a limited screen space for a user with limited visual attention space, the system needs to select the most interesting or important stream(s) for the user at any given time point. For example, for a distant learning application, the person who is speaking is generally assumed to be of more interest to remote viewers, so the camera video showing his talking head should be given higher priority over videos of non-active persons in the stream selection decision.
- **User Customization**: the question of whether one visual stream is more important or interesting than another is subjective to each user's judgment, and different users may have different preferences among the same set of visual streams. Therefore, it is crucial for any visual space management solution to allow each user to customize his/her selection of visual streams for screen rendering.
- **Visual Space Utilization**: since the visual space is very limited relative to the amount of content available, it is very important to utilize the available visual space to present important/interesting streams.
- **Automation**: although a user would want the visual streams presented to him in a customized way, the system would not be useful if the user has to interact with the user/system interface very frequently, which would distract the user from his main task.

- **Visual Quality**: although we cannot expect the computer "directing" automatically various visual streams to achieve the same level of visual quality as "human directing" in film making, there are certain criteria that qualify what is a visually pleasing video production. For example, it is a common practice in the movie industry that the visual streams on screen should not change too frequently so that the audience has enough time to digest the visual message, yet there should also be periodic changes so that the rendered program is engaging to watch.
- **Flexibility:** the solution to visual space management should be flexible so that it can be applied to different applications and different resource availabilities (such as number of streams or screen resolution).

We will first talk about why existing solutions fail to satisfy all of these requirements in Section 1.2, and then introduce our solution in Section 1.3.

1.2 Existing Solutions

The problem of managing multiple visual streams has occurred in many different applications, although we are the first to propose the umbrella model of the visual space management for all of them. In this section, we will classify existing solutions according to the approach they adopt in solving this problem: whether they choose to allocate the screen space to all the visual streams or some of them, and for the latter case, whether the stream selection and screen layout arrangement decisions are made manually (by the viewer or by professionals) or automatically.

1.2.1 Show All Visual Streams to Audience

Systems in this category (e.g. Virtual Auditorium [17], Digital Amphitheater [21], Access Grid [1]) adopt the simple and straightforward approach of presenting all visual streams to the user. For example, with the distributed learning system Virtual Auditorium [17], video windows showing all remote students are tiled on a large display wall in multiple columns and rows. The user (viewer) does not change the layout of these video windows on the screen, but chooses which visual stream their eyes focus on and simply ignores the other video windows on the screen.

One of the key advantages of such a “show all” solution is that the users have access to all the available visual streams related to the current application at any time. Therefore, the users have the maximal freedom to choose which visual stream to look at, and it is very easy for the user to switch his visual attention to another visual stream. The other key advantage is its simplicity: it is straightforward to implement such a solution, and the users can easily understand how all the visual streams are managed by the system and arranged on the screen.

Nevertheless, there are some inevitable limitations of such a “show all” approach. First, in a real world application, the total screen resolution required by all visual streams to be shown at normal quality (not to mention high visual quality) can easily exceed the total amount of available screen space (determined by the available display device and the distance between the user and the display device). Therefore, oftentimes the visual streams will have to be scaled down and squeezed into the limited screen space, which fails to provide the desirable visual quality that is essential to viewer experience, and also leads to scalability issues since the total number of visual streams is limited. On the other hand, if super resolution display walls are

used, then the cost of such systems will prevent it from being widely used. Second, since at any given time point the user's visual attention would be focusing on a single visual stream, the presence of all the other streams on the screen not only wastes valuable screen space, but also distracts the user from watching the stream he is interested in[1]. Third, while the user is focusing on one visual stream, he would have to rely on his/her peripheral vision to discover interesting events in other visual streams. This is not a reliable solution, and the problem of "missing event" may occur. For example, suppose a soccer game is captured by 30 cameras from different angles and distances, and all the 30 video streams are played on a large display wall. If a viewer is most interested in the camera showing the frontal view of a running player, it would be extremely difficult for him to browse through all the 30 video streams and find the best view he wants. Finally, from a social perspective, in many distributed conferencing scenarios (such as in distant learning applications), a participant may feel uncomfortable to know that he/she is always shown on the screen for remote participants even when he/she is not speaking or actively involved in the conference.

1.2.2 Show Streams Selected By Viewer

One step forward from the "show all" approach, many systems (e.g. ConferenceXP [4], OpenTV [8], Microsoft IP TV [5]) adopt a "viewer selection" approach, where all the visual streams are presented to the user in small tiled windows, and the user chooses which one to watch in high resolution (in a larger window or in full screen mode) by interacting with the graphic interface. For example, with the ConferenceXP Research Platform [4], the display

[1] Note that all the visual streams may be continuously changing and so visually distractive

windows of all the visual streams, such as cameras videos, the PowerPoint presentation, WWW browsing, whiteboard, etc., are tiled one by one on the screen on initialization. Then the user can manually arrange these windows on the screen, such as resizing a window or placing it at a different location on the screen, and also minimizing/maximizing a particular window. For another example, many leading Interactive TV (iTV) service providers, such as OpenTV [8] and Microsoft IP TV [5], support multiple view angels for sports game broadcast: all view angle choices are presented in split-screen mode at system start-up, and the viewer can pick any one video to watch in full screen with the remote control. In the Gaia system [37], Roman proposed an application framework where the content presentation is decoupled from the content modeler and controller, and the viewer can assign what content is to be shown on which display devices using a script language.

One of the key advantages of such a “viewer selection” approach is that high visual quality can be achieved even with ordinary display devices. No matter how many visual streams are involved in a session, the user can always choose to allocate the whole screen space to one or a few visual streams that interest him/her most at any time. In addition, the user has complete freedom in customizing how the visual stream windows are arranged on the screen, so each user can tailor the screen output on his/her display device based on personal preference.

However, such an approach also has two obvious limitations. First, the user interest may change as the content of different visual streams keep changing over time, so the user may have to frequently manually adjust the stream selection and the screen layout of selected streams, which would distract him/her from the main task (e.g. enjoying a show or collaborating with remote participants). Second, the problem of “missing event” is aggregated,

since when the user allocates all the screen space for one or a few specific visual streams, he/she does not have a clue about what is happening in other hidden streams, so it is more likely that he/she will miss some interesting shots that he would like to watch.

1.2.3 Show Streams Selected By Professional Human Editor

To achieve superior audience experience, a widely adopted approach is to hire human operators to *direct* the stream selection and screen layout arrangement process. For example, in the ePresence system [13] for webcasting based distant learning, a "human moderator" is hired to manually select a particular camera video stream to send to remote participants. Also, in the TV/film industry, professional human editors will decide which camera video to put on air for the audiences during live broadcast.

Such an approach has two key advantages. First, the users are saved from any distractive interactions, so that they can focus on their main task. Second, much higher video production quality is achieved: not only because the visual quality is higher, but also because the human editor can apply many videography rules in the directing process. For example, he may limit the minimum and maximum time for a particular camera view to be shown, and he may follow particular "idioms" [12] to deliver the visual message to the audiences. For instance, as documented in [12], when filming a scene of two person conversation, it is best to show an overview shot of the two persons' faces, followed by alternating views of each person's face and the overview shot.

Although such an approach has been well accepted by both the filming industry and TV audiences, it still has some drawbacks. First, the editing crew is often not affordable in many applications. For example, as distant learning becomes widely used, we cannot expect a human

editor to be hired for each class to conduct video editing. Second, the human editor makes the judgment of which visual stream to show on air based on both what he believes is important and interesting to the audiences. This may be different from the audiences' actual interest. Considering that different audiences may have different interests, the human editor can never satisfy everyone as long as he is producing a single video program that is presented to all audiences. Third, the human editor is still a human being, and often he still needs to watch all visual streams to find the one that is interesting. Although he may have been trained to be more proficient in predicting and discovering interesting shots, he may still miss some interest moments that could have been presented to the audiences due to fatigue.

1.2.4 Show Streams Selected Automatically

To avoid the cost of human operators, several systems (e.g. [23] [37] [34] [2]) have been proposed to use an automatic stream switching or "virtual directing" approach: to have the computer automatically select which visual stream(s) to put on the screen. These systems can be further classified into the following two categories based on how the automatic switching decision is made (automata based vs. score based).

1.2.4.1 Automata Based Automatic Stream Selection

Example systems in this category are iCam [37] and AutoAuditorium [2]. In the lecture capturing system iCam [37], Rui et al. propose to switch between the audience camera and lecturer camera based on a finite state machine. The states of the automaton represent the current views of each camera (such as lecturer camera's "zoom-in view" and "global view") and which camera is selected to put "on air" (lecturer camera or audience camera), while the transitions between the states represent various events such as detection of an audience speaker

or timer expiration. Similarly, in AutoAuditorium [2], a predefined automaton is used to direct the switching between showing the lecturer camera, the presentation slide and the global view of front stage of the lecture room.

There are three key advantages of such systems. First, the cost of human editor is saved, which greatly decreases the total cost of deployment of such systems. Second, high visual quality is preserved since the screen space is often allocated to one or a few visual streams. Third, the users are not distracted from their main tasks because the system works on its own.

However, there are still several aspects with such automata-based automatic switching approaches that can be further improved. First, the interest model for viewers is often fixed. For example, it is always assumed that when some person begins to talk, the camera view of that person's face should be switched on screen as soon as possible. However, a viewer may want to stick to other views, or he may be more interested in watching the reaction of others rather than the speaking person. Such a fixed interest model prevents customization based on the preference of a particular user, and it does not support more complex user interest variations in real-world applications. Second, the finite state machine needs to be manually authored by the system administrator, typically in some scripting language, which makes it very difficult for the system administrator or the viewers to modify the default stream selection rules. For example, iCam's automaton is specified as a cascade of decision rules, which are interdependent to each other. To modify the system to support one more camera, or to support a camera with different states and visual events, the whole automaton needs to be rewritten. This whole process is tedious and error prone.

1.2.4.2 Score Based Automatic Stream Selection

The other category of automatic visual stream selection solutions (e.g. [23] [34]) switch between different visual streams based on their *scores*. The score generally represents a measurement of how much a viewer desires to watch a particular visual stream, and the score for each stream will change over time based on the change of information delivered in each visual stream and also the change in user interest. By comparing the scores, the system would know which streams are the most desirable. In different systems, the score is calculated in different ways. For example, in Multiple Perspective Interactive Video [23], interesting objects/people are tracked in multiple camera views to generate a world model. A "best view" is then computed based on properties of these objects/people, such as "object/people occlusion degree" or "distance between object/people and the camera", etc. On the other hand, in [34], multiple cameras monitor the same room from different angles, and the "best view" is selected based on how clearly the face of the human subject is captured from each camera view.

Besides the three key advantages described above for automatic stream switching solutions, there is another unique strongpoint for such score-based systems: the user is saved from authoring the complicated finite state automata, since the system does not maintain any state information and only relies on the current scores of all visual streams to make rendering decisions.

However, previous score-based solutions share the same limitation with the automata based approach in that the automatic rendering output is uniformly defined for all users, so it is not possible to customize the output based on each user's preference. In addition, the lack of state information (which visual streams are showing what, which visual streams have been

selected for how long) limits the rendering capability of systems. For example, it is not possible to switch from a high scoring stream to a low scoring one, and it is not possible to switch to a particular stream based on the current rendering state of the system according to some predefined idiom. As we will describe in the dissertation, we adopt a hybrid approach that allows the user to adapt the default score setting dynamically and translate that information to a state automaton in runtime, which solves the limitations of these existing solutions.

Overall, many researchers have explored the solution space for visual space management (VSM) in different directions. Although they have achieved great success in building solid systems that are widely used, they still fall short of the goal of providing high quality, interest driven, customized and automatic visual stream selection and screen rendering results. This is the driving motivation behind the presented work, as will be introduced in Section 1.3.

1.3 Overview of MyView Framework

In this work, we present an innovative framework for visual space management called "MyView" that is *customized*, *automatic*, and *user interest driven*. Each user (or each group of users sharing the same display devices) can determine a unique stream selection and rendering result based on his/her own preference. In other words, for the same set of visual streams, the stream selection result may be completely different for users with different preferences. Our solution does not require users to repeatedly interact with the system during the runtime, yet it can still make the "right" decision on which visual streams to present to a user and how their display windows are arranged within the screen space. Specifically, we abstract each visual stream with a "shot-based" stream model, so that we are able to learn and predict a user's

interest among all visual streams by allowing the user to customize the weight of different streams' shots.

To achieve these unique features, our visual space management framework includes the following 3 phases:

- *Visual Stream Source Management*

  Since we are working with real-time continuous visual streams carrying visual information that is changing over time, we need to maintain updated knowledge about each visual stream so that we can predict a user's evaluation of each stream in terms of interestingness and importance. To achieve this goal, we adopt the shot-based visual stream model that abstracts each visual stream as a sequence of changing shot types, and the score-based user interest model that continuously predicts the current user interest.

- *Visual Stream Selection Management*

  The visual stream selection management module is the core component of our framework. It maintains user interest and makes the decision of *which visual streams are selected* for rendering on screen. This decision may change over time due to changes in the visual information carried with the visual streams, changes in user preference, or changes in available screen space.

  The visual stream selection management module works according to a timed automaton that maintains the current "state" of all the visual streams and the stream selection, and it also specifies how the state is changed based on changes in visual information and other constraints. The timed automaton is generated automatically from the user interest model.

- *Screen Space Management*

The screen space manager serves two purposes. First, it keeps an inventory of the currently available screen spaces. Second, given the set of selected visual streams decided by the stream selection manager, the screen space manager needs to calculate what screen layout to use to render these streams on the screen to fully utilize the given screen real estate.

The major contributions of the MyView visual space management framework are the following.

- **User Interest Modeling**: the visual space manager has a unique *user interest model* that is both easy to understand by users and functionally complete to express detailed viewer preference for stream selection. It represents each visual stream with a "shot based visual stream model" and adopts a four level "scoring scheme" for users to express their preference in a straightforward and generic way.
- **Timed Automaton-based Stream Selection**: we combine score-based and automata-based automatic stream selection approaches by translating the changing user interest and visual stream content into a timed automaton that makes the runtime stream selection decision. The unique feature of our framework is that the automaton is *automatically* translated from the user interest model, which makes the system much more flexible in runtime and allows for formal verification of desired properties. The functional correctness of the timed automata is verified using the UPPAAL [10] verification toolkit.
- **Screen Layout Calculation Algorithm**: existing solutions usually deal with a single fixed screen space and rely on a set of predefined screen layout patterns to arrange multiple visual stream windows on the screen. Our *screen layout management* is much more

flexible. It relies on a real-time screen layout calculation algorithm that computes the best screen layout pattern in real time to work with different number of selected visual streams and different number of displays of various resolutions.

- **Implementation and Validation**: to study and verify the usability of our solution, we have implemented a presentation broadcasting and recording system on top of the ConferenceXP Research Platform. It is a fully functional system that supports PowerPoint presentation, video/audio conferencing, whiteboard sharing, WWW co-browsing, etc. It has been used to validate the usability of MyView via a user study.

1.4 Thesis Outline

The rest of this thesis is organized in the following way. In Chapter 2, we present the model of visual space management for multiple visual streams and lay out the key assumptions behind our solution. In Chapter 3 we introduce three key background concepts of the shot based visual stream model, the timed automaton based stream selection process and user experience analysis that serve as the foundation of our work. An overview of MyView framework is given in Chapter 4, where the whole visual space management process is decoupled into three tasks and two stages. After that, the two key algorithms are presented in details: the timed automaton based visual stream selection algorithm (Chapter 5) and the screen space calculation algorithm (Chapter 6). Chapter 7 discusses some alternative solution frameworks and the tradeoffs we have made to arrive at our current solution. Chapter 8 presents the evaluation results of both the computer simulation based verification experiment and a mid-scale user study using an

example lecture recording/playback system we have built. Chapter 9 discusses related work and finally Chapter 10 summarizes this thesis and discusses future work.

Chapter 2. PROBLEM STATEMENT

2.1 Overview of Visual Space Management Architecture

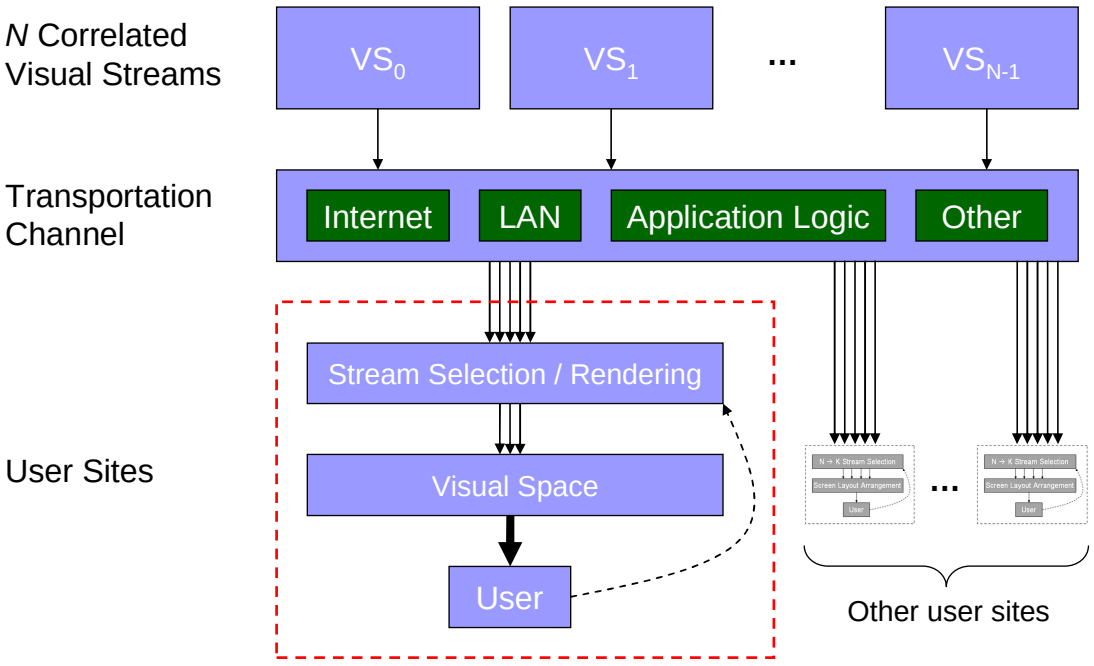

Figure 2. Overview of Visual Space Management Architecture

In this chapter, we introduce the visual space management (VSM) components in a multi-stream environment. An architecture overview is shown in Figure 2. *N* visual streams VS_0 to VS_{N-1} carry the visual information to be presented to the user. They are all transported to each user site via various kinds of transportation channels. At each user site, K ($0 < K < N$) of these *N* visual streams are selected for display based on user interest. The screen layout of the *K* output windows is calculated based on the properties of the visual streams as well as the total available screen space. While watching the selected visual streams on the display devices, the user may change the setting among the visual streams.

Visual space management is a complex problem with a suite of sub-problems, and it can be studied in many different ways. With MyView, we choose to focus on the visual stream selection and screen layout part and include the viewer's preference into the loop. Below we will describe each key component of the VSM architecture in details.

2.2 Visual Stream Model

In our framework, we adopt the following definition of a visual stream: a visual stream is any data stream that carries visual information to be presented to viewers on display devices, together with any metadata. Examples of visual streams can be image sequences from cameras, PowerPoint slideshow, web browsing session, etc. Note that the metadata can be the audio track accompanying a video stream, or some ink strokes on a PowerPoint slide. They are not directly related to screen display of visual streams, but they can be used to define semantic events that help the computer system to discover interesting segments of a visual stream. One of the unique characteristics of the visual space management is that it works with *multiple* visual streams, which cause the screen space and user visual attention space to become the bottleneck of visual information flow. As we have already given some examples of visual streams in video conferencing and other scenarios, they have the following common characteristics. First, they are continuous data flows carrying visual information, and the change in the visual information carried by each stream leads to changing user interest and importance level of that stream dynamically. Second, these visual streams are correlated in that they are to be concurrently presented to the user according to a synchronized rendering clock, and they present complementary visual information (similar but not the same) to a user for a common subject. Third, each visual stream requires a certain amount of screen space for their

presentation, and there is generally a lower bound on the screen resolution for the stream's visual information to be effectively presented to the users.

Therefore, it has become a challenging problem to abstract a high level visual stream model for visual space management. We have assembled a list of desirable properties for a good visual stream model as follows:

- **Easy to understand by users**

  The stream model is closely related to viewer interest representation. The more a user understands how the system models each visual stream and user interest knowledge, the better the system "understands" this user in his/her visual stream preferences.

- **Reflecting change in visual stream content that may lead to change of user interest**

  The visual stream model should be able to reflect the change in visual information the stream carries, but the change should be limited to those that are related to change of user interest. This is because normally the visual stream's information is changing all the time, but not all the changes will lead to a change in user interest. For example, for a camera video stream showing a person's head and shoulder, interesting changes may be the event that this person begins to talk or smile. On the other hand, if that person does not do anything special, his upper body may still move forward and backward slightly, but such changes in the video are not going to change the user interest level in this stream.

- **Supporting both instantaneous events and continuous events**

  Sometimes the change in visual information carried by a visual stream is triggered by an instantaneous event that occurs only at a time point, such as "a slide change in a PowerPoint slideshow", or "a stroke on the whiteboard"; while sometimes the event will

continue for a length of time, such as “someone is talking”, or “the camera is showing a close-up view of a person”. The visual stream model we define must be able to support both types of events.

- **Generic and widely applicable to any visual streams**

  Finally, we target a generic solution that works with the management of any type of visual streams, so the visual stream model should not be confined to specific types of streams. For example, the “talking” event is obviously a good indicator for user interest in camera videos, but the visual stream model can not include a “talking state” for every visual stream, because many streams (such as the PowerPoint slideshow) do not have a corresponding notion of “talking”. Therefore, the model we look for should be very generic and flexible and even support those potential types of streams that do not exist today but will be invented in future applications.

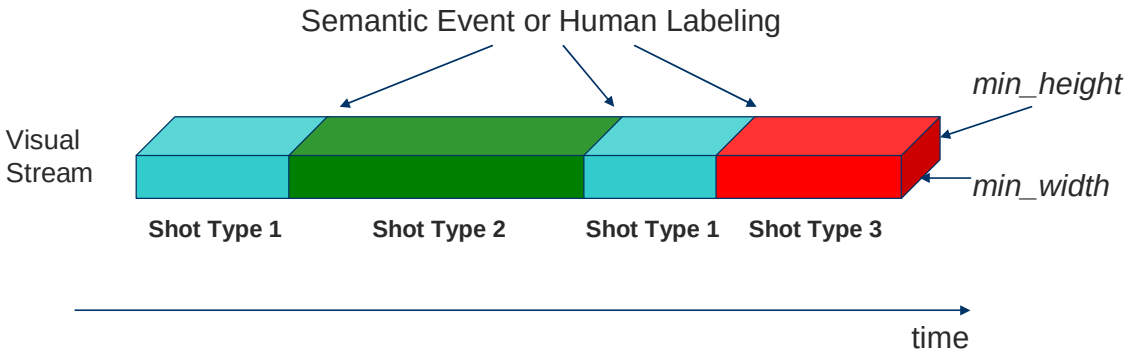

Figure 3. Shot-Based Visual Stream Model

With these requirements in mind, we adopt a “shot based visual stream model”. An overview of the shot-based visual stream model is shown in Figure 3 above. Specifically, each visual stream VS_i is represented by the following parameters:

- *Current_Shot_Type_i*: the metadata representing the current shot type of VS_i at the current time point. We adopt the traditional *shot based* visual stream representation, and each visual stream has M_i types of shots. At any time point, each stream's current shot is one of the *Mi* types, and the shot type may change over time. The shot definition is based on the key assumption that the user's interest in a visual stream should remain the same during each shot, and there can be three ways to define and detect a shot type change in a visual stream.

  - *Shots Defined By Automatically Detected Events*

    Such shots are defined by any automatically detected visual/audio event(s) that may lead to a change in user interest for that stream. For example, let us consider a video camera showing the head and shoulder of a person. If we assume that the user interest in selecting this camera stream (over other visual streams) will be determined by whether the person is talking, then we can define the camera stream to have two types of shots: "talking shot" and "silent shot". When the computer detects that the person starts to talk, the stream's shot type is changed to "talking shot", and the user interest level remains the same until the person stops talking and the stream changes to the "silent shot".

  - *Shots Defined By Automatic Timer Events*

    Such shots represent the change in user interest that is triggered by elapse of time. For example, if a user draws a new stroke on the whiteboard, then the whiteboard stream carries new information, and it may be labeled as the "new stroke shot". However, if a certain among of time (such as 120 seconds) pass by and there has been no change to

the information carried by the whiteboard, then the user's interest in that whiteboard stream will be reduced, so we can label that stream to a "stale shot".

- *Shots Defined By Human Editor*

  The shot type may also be determined by human editors if the cost of hiring them can be justified by the wide usage of the shot type information, and the advance of automatic event detection may replace the human editor in the future. For example, for a soccer game broadcast for millions of audiences, each camera stream can be labeled by a dedicated human editor as a specific semantic shot type. For example, if a camera tracks a particular player, then the shot type can be "Dash", "Feint", "Pass", "Running without ball", etc. Such information will give the audience the choice to customize what type of shots he wants to watch at any given time.

- *RC_i* = [*Min_Width_i, Min_Height_i*]: the metadata representing the resolution constraint for VS_i. We assume there is a minimum width and a minimum height for a visual stream to be rendered on the screen, and a visual stream should always be allocated a screen space that satisfies these constraints for display. For example, for a web browsing application interface, the browser window needs to be at least 600 pixels wide and 300 pixels high for the user to be able to comfortably read through the content of the webpage, so this constraint needs to be considered in the screen space allocation process.

Note that the model does not limit the source of visual streams. The *N* visual streams may come from real time content generation sources such as video cameras, remote content streaming servers or user interface of applications, or from storage sources such as video files

on the hard disk, or some combination of them. So long as all the *N* visual streams need to be presented to the user in a concurrent fashion, and change in the visual information carried in these streams leads to user interest change in real time, our solution will be able to work with them.

2.3 Transportation Channel Model

Depending on the type of visual stream sources, there can be many different types of transportation channels, such as the Internet, the Local Area Network (LAN), or the application logic, etc. There can be many interesting and challenging problems associated with simultaneously transmitting *N* visual streams and the associated metadata, such as the encoding method for data and metadata, the transmission protocol between sender and receiver, transmission delay and jitter, synchronization between the *N* visual streams, network bandwidth limit and transmission rate control, data Multicast at IP level or at the application level, compression and error concealment for the transmission, security, etc. All these problems are valid but not the focus of our research, and we assume an *ideal transportation channel* with the following properties:

- **Synchronization**: we assume that all *N* visual streams and the change of shot type of any visual stream will be transported to all user sites at the same time.
- **Abundant Bandwidth**: we assume that three is abundant bandwidth so that all *N* visual streams can be sent to each user within an acceptable delay.

Such assumptions are already valid for many applications today, such as video/data conferencing over the Multicast enabled high bandwidth network Internet2. For other applications with more constraints on the transportation channel, future research will need to

be conducted to work with network issues while the core idea and algorithms of the thesis solution will still apply.

2.4 Visual Space Model

Visual space management depends on how much screen space is available, which depends on several factors.

- First, there can be many types of display devices, such as hand held devices (cell phones, PDAs and iPAQs), laptop screens, Tablet PC screens, PC monitors, projector screens, plasma displays, TV monitors and others. Each display device may have a different resolution that ranges from a lower resolution such as 320x240 pixels to a higher one such as 2000x1600 pixels.
- Second, there can be more than one display device at each user site. Note that having two low resolution display devices is not the same as one high resolution display, because normally one visual stream cannot be split and shown on two separate display devices that are not physically collated with no space in between.
- Third, the user may want to limit the screen space allocated to an application for two reasons. First, the user's visual attention space is limited, so even if he has 10 display devices, he can normally attend to one or two of them at the same time at most. Second, the user may want to save some of the total available screen space for other purposes. For example, when the PC is connected to two displays, the user may just want to use one monitor for the multi-stream rendering application and preserve the other display for other applications.

To represent these variations in the screen space, we adopt an abstraction model for the heterogeneous display devices called *screen area*. A screen area is a rectangle region on the screen with a fixed resolution (*width* * *height*). During the setup process of the application, the user will be prompted to specify *M* screen areas that he wants to allocate to the visual stream rendering: SA_0 to SA_{M-1}. There is no constraint on the number or size of the screen areas, and they can be on the same or different physical display device. To help the screen layout calculation, in the case of multiple screen areas, the user will need to specify the *M* screen areas in the order of decreasing priority (the first screen area is to be used for displaying the most interesting visual streams). That is, for any indexes $i < j$, screen area SA_i will be considered for hosting visual streams before SA_j. With the help of the screen area model, the visual space management solution can ignore the actual physical display devices used for rendering, and focus on placing the *K* selected visual streams onto the specified screen areas. An example screen area definition is shown in Figure 4 below, with two display devices and 3 user-defined screen areas, and SA_1 is of highest priority

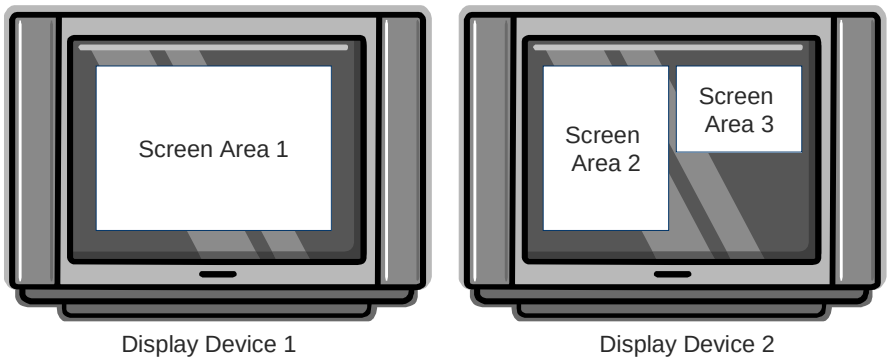

Figure 4. Illustration of Screen Area Model

With the screen area model defining how much visual space we have, we also need to specify how these screen areas are allocated to each of the *K* selected visual streams. Therefore, we use *screen layout pattern* to represent a specific segmentation of the screen areas

into *K* "subwindows", where each stream is shown in one of the subwindows. Formally, a screen layout pattern *SLP* = [$Subwindow_0$, $Subwindow_1$, … $Subwindow_{K-1}$], where $Subwindow_i$ = [SA_index, $left_i$, top_i, $width_i$, $height_i$], representing the index of the screen area, the coordinate of the left/top corner of the subwindow relative to the left/top corner of the screen area, and the width and height of the subwindow. For example, for the screen layout pattern consisting of 3 subwindows (as shown in Figure 5), it can be represented as SLP = [[0, 0, 0, 600, 300], [0, 0, 300, 300, 200], [0, 300, 300, 300, 200]].

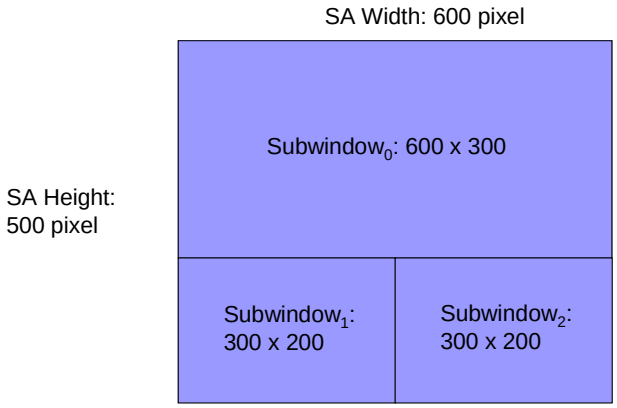

Figure 5. Example Screen Layout Pattern with Three Subwindows

When a set of visual streams are selected for screen display, a screen layout pattern will need to be generated to map the streams to the subwindows on the screen areas. An example of the mapping is shown in Figure 6 below. Three visual streams are selected [VS3, VS1, VS2] for screen display, and there are two display devices with two screen areas SA1 and SA2 defined on them. A screen layout pattern is created with one subwindow on SA1 and two subwindows splitting SA2 vertically.

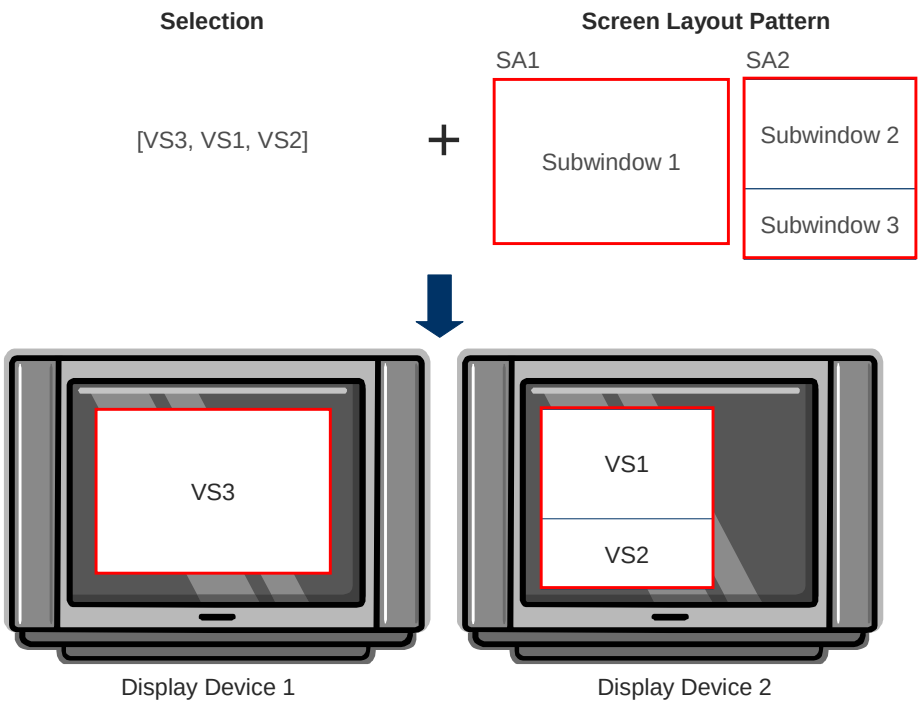

Figure 6. Mapping Stream Selection to Screen Layout Pattern

2.5 User Interest Model

A user in the problem model can be a single person or a group of people sharing the same display devices and preference among the visual streams. Although there may be difference in the user interest among the group of users, we assume they will come to a common preference for the different shots of all the visual streams since they are sharing the same display output. With this assumption, the case of a group of users reduces to the single user case, and our discussion below will only focus on the single user case.

We assume each user has a dedicated *user site* with one or multiple display devices, and the stream selection and screen layout will be based on the user's (changing) interest. User

interest is a very subjective variable to quantify and predict, and we designed an innovative *score-based user interest model* that represents each user's preference by the following parameters:

A. Interest/Importance Score Level

To explain the thoughts behind the score level based scheme, let's imagine an average person watching a fashion show on TV at home. Suppose there are several TV content providers broadcasting this event at the same time, such as ABC, FOX and NBC, and each TV provider employs a crew of cameraman to capture the same show from multiple view angles and multiple sites. Suppose that at a specific moment, ABC's camera is showing the models at the back stage, while FOX's camera is tracking the frontal view of the models on stage, and NBC's camera is following the model's rear view. In such circumstances, it is natural for the viewer to say "I think the ABC channel showing the back stage is more interesting than the frontal view of models shown on FOX channel, and both of them are more interesting than NBC's rear view.

The interest/importance score level models exactly this kind of user preference among all the visual streams, and it is built on top of the "shot-based visual stream model" introduced in Section 2.2. Specifically, for each visual stream's each *shot type*, a score is assigned to represent the user's interest level in that shot. For example, if visual stream VS1's shot type X is assigned a higher score than visual stream V2's shot type Y, it indicates that when these two streams are in these two shots respectively at the same time, then the user prefers to watch VS1. Although this idea of using score level to represent user preference is pretty straightforward, there are two specific questions that have to be answered.

1. *Score assignment user interface (UI).* The interface has to be designed in a way that the user can easily change the score for a specific shot type, and it should also clearly illustrate the relative ranking of the score among all the visual stream shots. Therefore, traditional user interface components, such as combo boxes or list boxes, will not be suitable (since the user cannot get a sense of the relative ranking at a glance).

2. *Range and granularity of scores.* We need to decide what are the range and granularity of scores, which directly determine the ease of understanding and use of the score panel. For example, the score range for each shot type may be any integer between 1 and 100, or it can be any real number between 0 and 1.0. Such designs look reasonable at first, since it is only the relative score ranking that matters for representation of user preference. However, these designs have two unacceptable drawbacks. First, there will be too many score levels for the user to assign to a shot type and the user may get confused. For example, it does not mean anything to set shot type X's score to be 97 and shot type Y's score to be 89, and two simple score levels of 2 and 1 would do the same job with much less potential explanation and misunderstanding. Second, there is not a clear link between such score level assignment and the final stream selection decision. For example, suppose three visual streams are in the shot types of X, Y and Z, which are assigned the score levels of 90, 80 and 70. Neither the user nor the system would know which stream(s) should be selected for rendering – all three, or just the top ranking one?

The UI design for score assignment depends on the application and system designer, and we will present an example UI design in Chapter 7. For the question of scoring scheme, we have designed a *four level scoring scheme* as described below:

- **Score 3**: The top score level is called "*must show (high priority)*", which means that for any visual stream, if it is in a shot type assigned with this score level by the user, then that stream must be presented on the screen with higher priority in the screen layout calculation process. We will discuss more details of the screen layout calculation algorithm in Chapter 5, but now it suffices to know that visual stream shots with this score will definitely be displayed and also will be allocated a larger screen space (hence higher resolution) than those visual stream shots with lower score levels. This score level can be used by the user to specify the top priority stream shots.
- **Score 2**: The second top score level is called "*must show (low priority)*", which means that for any visual stream, if it is showing a shot with this score level at the current time *t*, then it must be presented on the screen, but it has lower priority in the screen layout calculation process. This score level can be used by the user to specify those shots that he is interested in watching, but not necessarily in high resolution.
- **Score 1**: The third score level is called "*show if room (show in rotation)*", which means that if there are no streams with a higher score level (score 3 or 2), then all visual streams showing shots assigned with this score level will be displayed on screen in rotation. That is, one stream will be rendered first, and then followed by another stream with the same score, etc. This score level is useful in the case when there are some shots that are not very interesting to watch or do not carry much new information, but if

they are the only choices, then they can be put on screen in rotation. Note that an alternative is to show all visual streams with score *1* together, but this is not scalable. It means if many streams are in a shot type with score *1* at the same time, then the visual quality of the screen output will be bad. Nevertheless, some application specific optimization may apply, such as show two streams at a time in rotation.

- **Score 0**: The bottom level is called "*do not show*", which clearly means that for any visual stream, if it is in a shot that is assigned the "do not show" score level, that stream shot should never be put on the screen (unless all streams' current shot types are assigned the "do not show" level, which is rare). This score level can be used for any stream shot type that the user does not want to watch, e.g., the user does not need to see the local camera video stream of himself on screen during a tele-conferencing

To better explain the meaning of the scores, we will give three examples below. Let us assume there are four visual streams *A, B, C* and *D*. For the first scenario, let us assume the current shot types of there streams have the following scores: *1* for *A*, *1* for *B*, *0* for *C*, and *1* for *D*. Then we know that *A, B, D* are of "*show in rotation*" scores, so they should be selected to display in rotation like $A \rightarrow B \rightarrow D \rightarrow A$, until some new change to the scores occurs. For the second scenario, let us assume the scores are: *1* for *A*, *1* for *B*, *0* for *C*, and *2* for *D*. Then we know stream *D* has a "*must show*" score, so it is selected for rendering. For the third case, let us assume the scores are: *3* for *A*, *1* for *B*, *0* for *C* and *2* for *D*. Then both stream *A* and *D* are selected because they both have a "*must show*" score ($\geqslant 2$), and *A* will be given higher priority in screen display (as will become clear in Chapter 6).

B. Minimum On Screen Time

For every shot type of each stream, the user can specify a "*minimum on screen time*", so that if it is put on screen, there will be a minimum time that the user wants the screen output to hold in the same way before changing to some other screen layout/display. This parameter serves two purposes. First, it is a common sense that the user needs some time to digest the visual information carried by a visual stream. For some streams, a short time period may be enough, such as a camera video showing a person's smiling face; for some other streams, a longer time may be needed, such as a new PowerPoint slide with a lot of text. Second, the minimum time helps to avoid too frequent changes of screen output, which some user may find annoying. Note that the minimum on screen time is associated with each shot type rather than each visual stream, since different shot types of the same stream may have different amount of new information or user interest. When there are multiple visual streams on screen, the minimum time for any screen layout change will be the *maximum* of the minimum on screen time of all the streams' current shot type.

C. Maximum on Screen Time

For each stream's each shot type, if it is assigned a "*show in rotation*" score level, then the user can also specify a maximum time that he wants to switch to some other visual stream with the same score of "show in rotation". This parameter helps to introduce more variations into the rendering result. Note that the maximum timer only has effect when there is no "must show with high priority" or "must show with low priority" visual streams.

D. Transition Idioms

In the film industry, an *idiom* is a predefined camera switching scheme for filming a particular type of scene. For example, one idiom may specify that "*to capture two persons' conversation, we need to present an overview shot for the two persons, and then switch between the two person's frontal face views*". Similarly, in a multi-stream application, there may be occasions when the user wants to switch from a stream shot with lower score to a higher scoring one. For example, suppose the user assigns a "must show with low priority" score to the "talking shot" of the lecturer's camera stream, and a "show in rotation" score to the "stale slide shot" of the PowerPoint slideshow stream. This means that by default the user is more interested in watching the lecturer's talking face then a slide if there has not been any slide change over a time period. However, the user may want the system to automatically switch back to the slide after the lecturer's speaking shot has been shown for a while because he wants to see the slides again to refresh his memory of the information on the slides. This represents a situation where a transition from a low scoring shot to a high scoring shot is desirable, and this cannot be supported with the score level and timing constraints alone. Therefore, we elect to support idioms in the following format: if the shot X of stream VS_i has been *on screen* for *m* seconds, then switch to show another visual stream VS_j.

To summarize, the benefits of using such a score-based user interest model are the following. First, it is easy to understand and straightforward to use, because it follows a natural way of thinking for users to compare between different types of shots from different visual streams. Second, the score levels are a generic concept and can be applied to any type of visual streams, which allows the system to compare different types of visual streams on a common ground. Without this level of indirection, it would be very hard for the user to directly express

the preference for a PowerPoint shot over a video camera shot. Third, the combination of scores, minimum/maximum on screen time and the transition idioms is very expressive for different users to specify completely different preference among the visual streams.

2.6 Conclusion

Visual space management in multi-stream environment is a very complicated problem in that the stream selection and rendering decision depend on several entities such as the user interest, the stream characteristics, the transportation channel, and the available screen space. We have designed a set of simple and innovative models. These models are concrete and realistic so that they capture the key challenges in the problem, yet generic and abstract so that it can be applied to any specific application scenarios. In the next chapter, we will discuss the background information that will help the readers to understand some of the important concepts behind the problem statement and our solutions.

Chapter 3. BACKGROUND

Before we introduce our solution, we will discuss three background concepts that serve as the foundations for our solution: (1) why we chose to use the concepts of shots and scores for modeling of visual streams and associated user interest; (2) why we adopted the timed automata model for the runtime stream selection process; and (3) what we can learn from traditional work on viewer's visual searching/sampling experience analysis in designing our solution.

3.1 Shots and Scores for Video Summarization

The question of "which video is more interesting" as asked in the visual space management problem is similar to the question of "which parts of a video is more interesting", and the latter is a classic problem studied in the area of video summarization, especially for generation of video highlights [28]. A video highlight is a short video skim consisting of the most interesting segments of a video clip. One widely used solution has been to segment a video stream into shots and then generate video highlights based on ranking the interest scores of shots. In fact, the intuition behind our solution to the visual space management problem for multiple visual streams is exactly to treat it as a "real time multi-stream summarization" problem and apply similar shot and score based methodology to solve it. Note that in our discussion, we will use the terms "video summarization" and "video highlight generation" interchangeably to express the process of generating a short summary that is composed of the most interesting segments of a video.

3.1.5 Single Stream Video Summarization

Given a video stream, video summarization generally follows three steps to generate a summary for it, as shown in Figure 7.

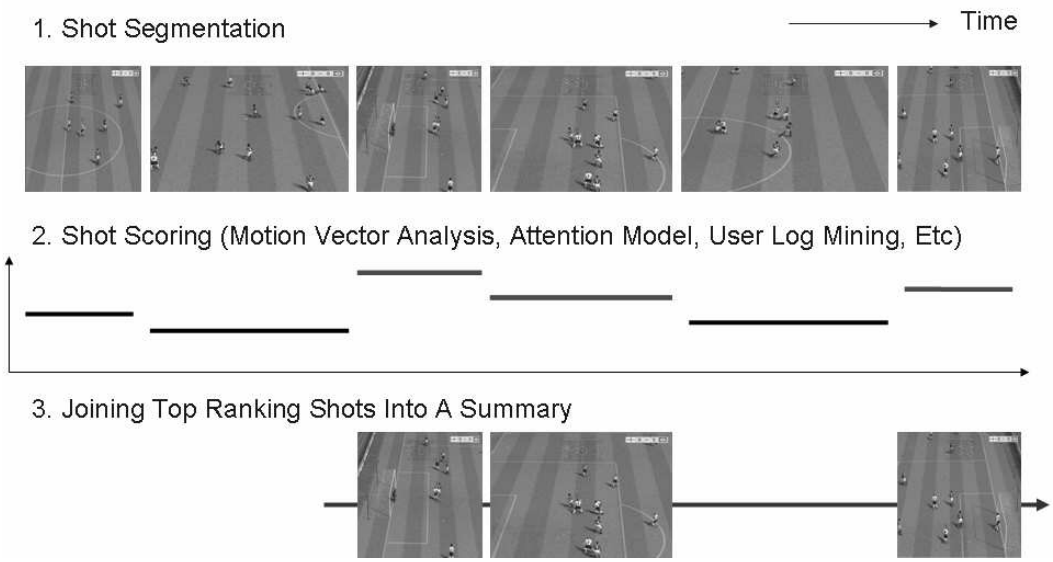

Figure 7. Single Stream Video Summarization

The first step is the shot segmentation, where the whole video clip is segmented into short independent video clips called “shots”. There can be many ways for conducting shot segmentation. For example, some low level features can be used to group a continuous sequence of video frames together, such as motion vectors or color component. On the other hand, high level semantic *events* can also be used to segment the video frames, such as the events that are associated with the movement of key objects/persons in each video stream, or events detected by mining user evaluation of the video frames via analyzing browsing log of users.

Once the shots are available, the second step is to assign an *interest score* to each shot and rank all the shots. The assignment often naturally comes from the specific shot segmentation method used in the first step. For example, if motion vector analysis is used for shot segmentation, then the motion vector strength can be used as the score for a shot; if the user log analysis is used for shot segmentation, then shots with higher user popularity will be assigned a higher score.

The third step is to select top ranking shots (by comparing the interest scores) and concatenate them into a short video summary. The question of exactly how many top ranking shots are selected can be affected by many factors, such as the desired length of the video summary, or a preset score threshold.

The key challenge in video highlight generation lies in the second step, i.e. how to define a good model for viewer's interest in each video shot. Many solutions have been proposed, generally falling into the following two categories: complete automatic solutions and semi-automatic solutions based on user interaction.

One example of complete automatic highlight generation is the VAbstract system [35] that automatically produces movie trailers. In this work, Pfeiffer et al. specify that a "good quality abstract" should contain *important objects/people* and *events* (such as dialog and scenes with strong motion). In [16], Chang et al. use a set of predefined *events* (such as "home run" and "within diamond play") to evaluate baseball video shots, while seven types of scene shots (such as "pitch view" and "catch close-up") are used to train a Hidden Markov Model that detects those events. In [31], Ma et al. propose a user attention model that evaluates each shot using attention attraction indicators, such as high motion, strong intensity contrast, face of

people, etc. Overall, such complete automatic approaches generate video highlights that capture the important objects/people and high motion/contrast scenes.

On the other hand, some semi-automatic solutions utilize the viewers' interactions as a feedback to the relevance/interest evaluation process. For example, in MediaMiner [41], Syeda-Mahmood et al. train a Hidden Markov Model for viewer interest as the viewers browse video clips using a traditional video player. In VideoAbstract [44], Toklu et al. propose a hybrid approach that automatically generates a video storyboard, and then utilizes user input to refine the storyboard into a meaningful video summary. In one of our previous work ([49]), the user's video browsing log is analyzed using link-analysis algorithms for web document retrieval to derive interest scores for video shots. Overall, such automatic systems gain more semantic understandings and higher level of user customization at the cost of requesting more user interactions.

3.1.6 Multi-Stream Summarization

The problem of visual space management can be regarded as how to generate a single visual stream summary for multiple visual streams. For example, suppose we have four video streams capturing different perspectives of the same soccer game, as shown in Figure 8, and we want to generate a single video as the final program by selecting segments from each of the four streams and concatenate them together. The intuition is that we can follow the same way of thinking in solving the single stream summarization problem: we can segment all four video streams into video shots using the same shot segmentation method, and assign scores to each shot in each video using the same method. This way, we get sequences of shots with scores that is *comparable between different streams*. Note that this cross stream comparability is very

important because it gives us the tool to select the most interesting/important shots among all the visual streams. With the help of these scores, we can identify a "top ranking shot" for each time point (step 2 in Figure 8), and then concatenate these shots from different visual streams into a single video program

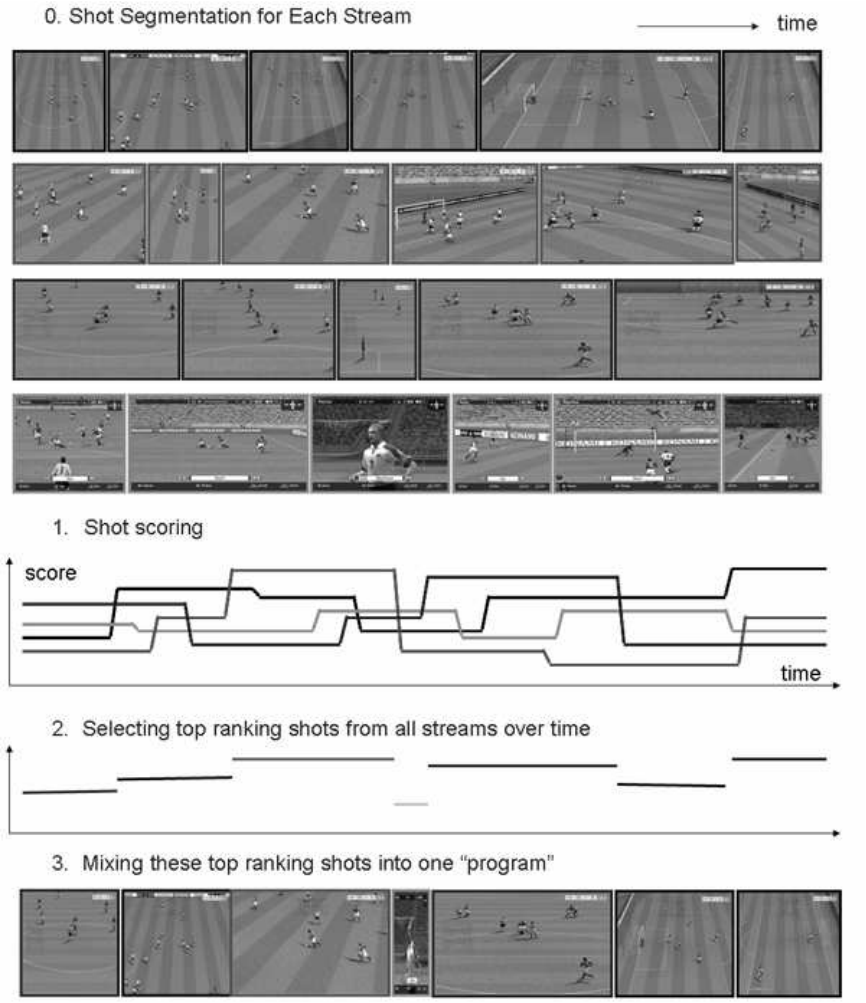

Figure 8. Intuition of Multi-Stream Summarization

Although such a "multi-stream video summarization" metaphor points out a promising direction to explore the solution to visual space management problem, there are some important questions we need to answer regarding the difference between these two problems

and additional restrictions and assumptions specific to the visual space management problem. Specifically, we need to consider the following aspects:

1. **How to conduct shot segmentation *in real time*?**

   Since the visual streams may be generated on-line, such as the case of video cameras, we cannot assume pre-knowledge about future incoming visual information carried in the visual streams. This puts a restriction on the type of shot segmentation method that no offline processing method may be used. Therefore, in our solution, we rely on automatically detected events and human editors to label the shot type of each stream.

2. **How to derive a good shot score for each visual stream shot?**

   Since we are aiming at high quality user preference-based customization, the interest score definition must well reflect the user's interest and it should be easily understood and customized during runtime. Therefore, we choose to learn the interest score for each type of shots from the user directly via a graphic user interface

3. **How to render multiple streams?**

   In visual space management, we are no longer producing one final video output as in the case of video summarization, so new algorithms are needed to calculate how to arrange the selected streams on screen based on their properties.

3.2 Timed Automaton Model

Timed automaton is one of the most widely used formal models for timed discrete event systems, and we will use it for both guiding runtime execution of stream selection and formal

verification of the correctness of this process. In this section, we will first discuss the reasons for choosing timed automata, and then introduce the specific formal description language UPPAAL [10] that is used in our solution.

3.2.7 Formal Models of Timed Discrete Event Systems

The process of visual stream selection and rendering depends on both timing information and the discrete events of changing shot types, so it can be formally modeled as a timed discrete event system. There are many different models for such systems, such as Timed Automata [11], Timed Petri Nets [33] and Discrete Time Process Algebra [14], etc. All these models are powerful languages for modeling the stream selection process, and we finally chose to adopt the timed automata model based on the following rationale:

As we will discuss in Chapter 4, one of the key components in our solution framework is the automatic generation of the stream selection model that can be used both to guide the runtime execution of the stream selection process as well as offline verification of the desirable properties and functional correctness of the process. To serve as the runtime execution engine, this model has to be simple to understand, easy to be generated automatically and consistent with existing thinking in related work. In this aspect, timed automaton [11] is a good choice because it is one of the simplest models among the choices, and it has already been used in most of existing solutions for multi-stream rendering applications. On the other hand, for the model verification functions, timed automata allow us to verify a wide range of properties of the stream selection algorithm, such as liveness (whether the process will become deadlock) and reachability (whether a specific state of the system will be reached after a finite sequence

of executions), and there exist excellent model verification tools to assist our analysis, such as UPPAAL [10].

For a brief introduction of the timed automaton, Figure 9 shows an example probabilistic timed automaton. A timed automaton is different from a normal automaton (finite state machine) only in that it has one or multiple *clocks* attached to it. Therefore, with a timed automaton, the state transition of a timed automaton depends not only on the firing conditions, but also the reading of the clocks. For example, in the automaton given in Figure 9, two clocks *x* and *y* are attached, and they are both reset to *0* during the initiation of the automaton. The transition from state s_1 to state s_2 has the condition "$y > 0$", which means when the time value of clock *y* passes *0*, the state transition from s_1 to s_2 will be triggered with a probability of *0.7*. In this work, we will only work with deterministic transitions, so the transition probabilities are always *1.0* and not explicitly marked.

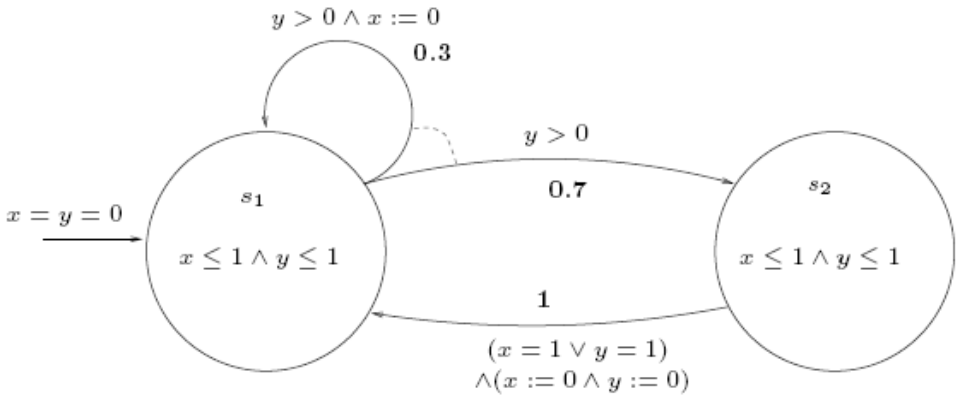

Figure 9. Example Timed Automaton

In the next subsection of the chapter, we will present more details of the UPPAAL tool environment for modeling and verification of discrete event systems, which will help the readers to understand the automata generation algorithm described in Chapter 5.

3.2.8 UPPAAL Introduction

UPPAAL is a tool for modeling, validation and verification of real-time systems model. It is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks (i.e. timed automata), communicating through channels and (or) shared data structures.

The UPPAAL tool consists of three main parts, a system editor, a simulator, and a verifier.

A. System Editor

The *system editor* is used to create and edit the system to be analyzed in a graphical user interface. A system description is defined by a set of process templates, some global declarations, process assignments and a system definition. Each automaton in UPPAAL is composed of one or multiple "*processes*", where each process is a finite state machine. The circles are the states of the automata (called "*locations*" in UPPAAL), and the edges between the locations are transitions of the automata (called "*transitions*" in UPPAAL).

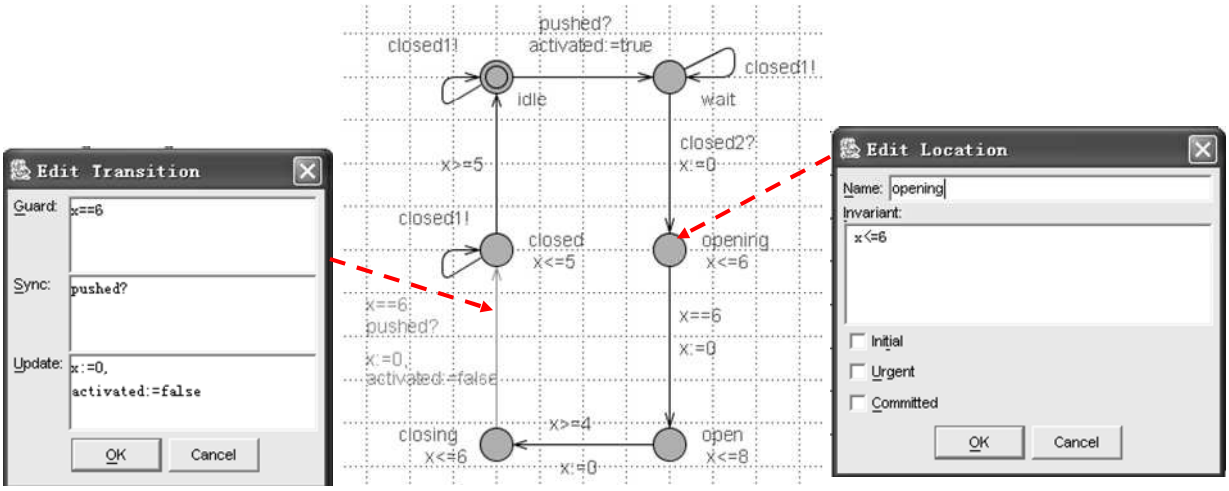

Figure 10. Example Automata in System Editor

For example, Figure 10 shows an example *timed automaton*. Each automaton may also have one or multiple "*clocks*" that simulate dense time (real numbers), such as "*x*" defined in the example. Each location has a *name*, and a set of "*invariants*" that specify the condition for the automaton to keep staying in a location. For example, in UPPAAL syntax, "$x \leq 6$" means that the automaton must leave the current location before the clock x passes 6 time units. Each transition is directional, and it has three properties: (1) The "*guard*" for a transition is any condition that must be satisfied for this transition to be enabled. For example, if the guard is "$x == 6$", this means the transition is enabled exactly at the time moment of 6 units. (2) The "*sync*" property is used for synchronization of transition execution between multiple automata. When sync is blank, it means this transition can fire independent from other automaton. When it is not blank, it is always specified using predefined "*channels*". Each channel has a channel name, such as "*pushed*", and it is used to specify the "*sync*" property of a pair of synchronized transitions with either a "*!*" or a "*?*" mark after it, such as "*pushed?*" and "*pushed!*". In the given example, unless there is another transition in another automaton that is enabled and has a

"*pushed!*" in its sync property, this transition will not be fired. (3) The "*update*" property specifies the effects of a transition firing, and it is always in the form of "*variable := new value*". For example, "*x := 0, activated := false*" means after this transition is taken, the clock "*x*" is reset to *0,* and the boolean variable "*activated*" is set to *false*.

B. System Simulator

The simulator is a validation tool which enables examination of dynamic execution of a system during early design (or modeling) stages and thus provides an inexpensive means of fault detection prior to verification by the model-checker. The simulator is also used to visualize executions (i.e. symbolic traces) generated by the verifier.

C. System Verifier

The verifier is to check invariant and liveness properties by on-the-fly exploration of the state-space of a system in terms of symbolic states represented by constraints. The verification tool in UPPAAL also provides a requirement specification editor for specifying and documenting the system requirements. We will discuss the simulator and verifiers more in the model evaluation section below.

3.3 Viewer Experience Analysis

Since the goal of this work is to improve user experience by customizable automatic visual stream management, it is essential to understand viewer's visual searching/sampling experiences and its implications on the design decisions of such a viewer-oriented visual space management system. Therefore, this section is dedicated to the analysis of user experience.

We can think of any visual stream management system as a helper that facilitates the users in their search for visual information. For example, in a lecture viewing scenario, ideally we would like the distributed users to have the same experience as if they are all physically collocated in the same lecture room: they can look at anything they want to, such as the presentation slides on the projector screen, the lecturer, or other students. However, because of the limitation of the bandwidth of information that can be conveyed through the visual window on a remote user's screen (either desktop monitor or projector screen), we cannot present all visual information to the users at the same time. Therefore, visual stream management can be viewed as a process that aims at predicting and presenting the set of visual information that the user could have wanted to look at if he/she was physically in the lecture room. This boils down to two questions: *how to select what information is presented* and *what is the best way to present this information*. To answer both questions would require an understanding of how a user allocates his *visual attention*. Therefore, in this section, we will first discuss the traditional classification of user behavior ("Goal-Driven" vs. "Stimulus-Driven"), and then introduce several classic *visual attention allocation models*. We will use lecture viewing process as an example to connect the visual searching theories with the practices of visual stream management. At the end of the discussion, we will analyze the unique characteristics of user experiences in lecture viewing and conclude some important implications on how to build a user friendly automatic visual stream management system.

3.3.9 User Experience Classifications

3.3.9.1 "Goal-Driven" vs. "Stimulus-Driven"

Tremendous existing work has been dedicated to the question of how visual attention is controlled, which can be classified into the following two categories.

With *active* or *top-down* attention, the visual attention allocation is said to be "goal-directed", meaning that it is under strict supervision according to the observer's goals. In [47], Yantis pointed out that there are ways for human subjects to intentionally direct their visual attention and select "where to look": *selection by location*, meaning that one can attend to a restricted region of space; *selection by feature*, meaning that one may seek objects with known visual properties (such as color or shape) but unknown location; *selection by segmented object*, meaning that one may select to look at one object and ignore another object even if the two objects occupy an overlapped two dimensional region.

On the other hand, with *passive* or *bottom-up* attention, the visual attention [20] is said to be "stimulus-driven", meaning that it is allocated based on the properties of the image being observed. There are generally two types of stimuli:

- Singleton:

  This refers to visually salient objects in contrast with the background image, such as objects with different colors, unique shapes or a special motion, etc. For example, it has been studied in [42] that subjects can easily search for a target shape (such as a diamond) within an array of distractors (such as circles).

- Visual Onset:

This refers to the abrupt onsets (consecutive appearance and then disappearance) of objects within a very short time. In [36], Remington et al conducted a series of experiments that ask the subject to make a speeded two-choice response to a letter that flushes in one of four boxes. Before the letter appears, distracting abrupt-onset visual stimulus (crosses) flushes around the target box (where the letter would appear) or around other locations (pure distraction). It has been discovered that even though the subjects know the locations of the distractor and the box where the letter is to appear, they are still affected by the distractors. The conclusion is that spatial visual attention can be involuntarily drawn to abrupt onsets despite the intention of the subjects to ignore them.

3.3.9.2 Analysis of Lecture Viewing Scenarios

It has been a general consensus that most visual search processes involve a combination of both goal-directed and stimulus-driven attention allocation. Connecting back to the scenario of a typical student attending a lecture in a classroom, his visual attention allocation can be reduced to the following key components based on the specific attention driver:

- *Goal-Directed*
  - When the student is reading the slides, his/her attention is subjectively directed at the fixed location on the screen;
  - When the student is watching the lecturer talking, his/her attention is directed at the face and gestures of the lecturer;

- *Stimulus-Driven*

- When there is a change in screen layout (such as the number of video/slide windows on screen, their relative position, their sizes), the student's visual attention will be attracted to it.
- When there is a new slide or the video window switches to show content from another camera (while remaining the same size and location), it will also attract the student's visual attention.

3.3.10 Visual Attention Allocation Models

3.3.10.1 Visual Information Acquisition Models

There has been a lot of existing work on modeling how the subject acquires visual information, and what factors will determine the effectiveness and bandwidth of visual information in-take of the subject. Models of visual information acquisition can generally be classified as "visual search" or "visual sampling" [45]. Visual search models emphasize locating a single target with a focus on the timing and accuracy of searching. They have been useful in a lot of applied domains such as searching maps and menus. In contrast, visual sampling models focus on supervisory control using primarily visual sampling and scanning. A typical example is the aviation task, where the pilot needs to monitor a series of dynamic processes (as opposed to a single static target), monitor critical events and allocate visual attention to various area of interest in different proportions. The focus moves from knowing *where to look* to *when to look where*.

One of the goals of these visual information acquisition models is to find an optimal strategy for visual attention allocation, so that the scarce resource of visual attention can be allocated to maximize some utility function or minimize some cost function. For example, one

extensively adopted model is the SEEV Model that characterizes the probability of the visual attention being allocated to a certain *area of interest* **A**: P(**A**) = s**S** - ef**EF** + (ex**EX**)(v**V**). Each term in capital and bold letter is a characteristic of a particular environment that is determined by (1) the physical properties of events (Salience = **S**), (2) the physical distance between a previously fixated and a current AOI, or the demands of concurrent tasks (Effort = **EF**), (3) an information-related measure of event expectancy (e.g., bandwidth, event rate; Expectancy = **EX**), and (4) an objective measure of the value (=**V**) of processing information at the AOI in question (or the cost of failing to attend there). The coefficients, *s*, *ef*, *ex*, and *v*, represent the relative influence (weight) of these four factors on human scanning.

3.3.10.2 Analysis of Lecture Viewing Scenarios

Although models such as the SEEV model have been successfully applied to wide range of applications (such as aviation training), we should not naively port them to the lecture viewing scenarios, which have the following unique characteristics:

- Students are accustomed to TV watching experiences, where they only need to pay attention to one or two streams most of the time. Therefore, we cannot expect them to be able to keep track of too many different data sources at the same time as the pilots do.
- Students are used to passively accept whatever is presented on the screen automatically (controlled by video switching professionals behind the scene), instead of actively manipulate the video streams.
- Students have pretty predictable events of interests for the video streams and slide streams, such as audio cues, gestures and slide transitions, which can potentially be leveraged to automate the video editing process.

Therefore, if we summarize all the visual attention allocation and visual searching/sampling models, we arrive at the following *important guidelines* for building a user friendly lecture broadcast system:

- The system should stick to familiar user experiences of "automatic" video switching and window management. We should try to minimize user involvement in setting up the system so that they can focus on the content of the lecture.
- We should aim at reducing user's visual searching efforts by trying to present new interesting information in using quick onset or salient colored borders so that it is easily and naturally captured by the user, and also try to present the same visual stream at the same screen location at the same size.
- We should try to minimize unnecessary distractions from the un-interesting visual streams, such as hiding un-interesting streams and avoid too frequent changes in screen layout or visual stream selection
- The system should be flexible enough so that its automatic stream selection reflects how much value a user assigns to different visual streams.

Chapter 4. MYVIEW FRAMEWORK

In this chapter, we present our solution to the visual space management problem based on the models described in Chapter 2. We will first present the MyView framework that divides visual stream management into three management entities (visual stream source management, visual stream selection management and visual stream rendering management), and then describe a concrete example of how such a model can be applied to a distributed class system. The next two chapters will be dedicated to two of the key algorithms (stream selection algorithm and screen layout calculation algorithm) in details.

4.1 Visual Stream Management In MyView

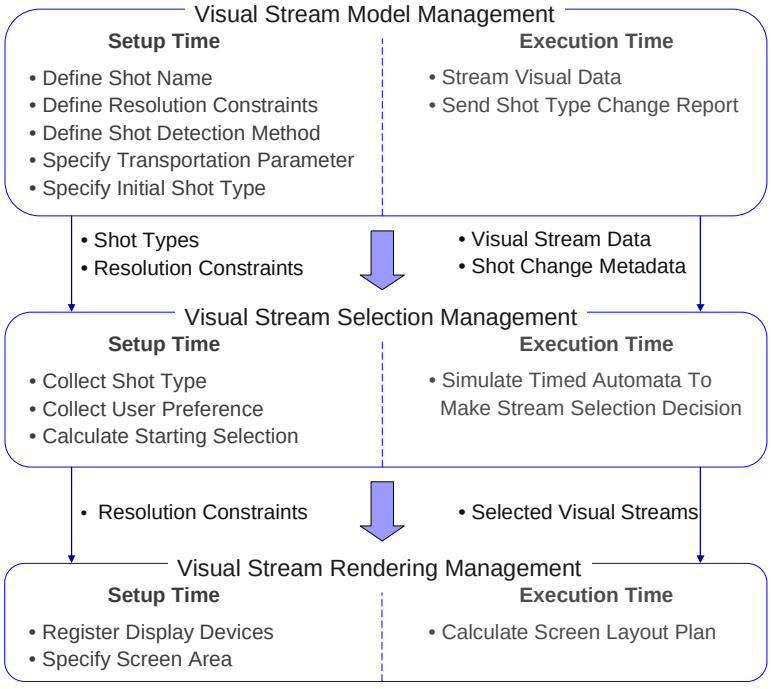

Figure 11. Overview of MyView Framework

Since the visual space management problem is a multi-dimensional problem that involves not only the end user sites but also the source of the visual streams, “MyView” is naturally divided into multiple smaller management tasks. As shown in Figure 11, “MyView” can be divided into three major *management entities* and two *management stages* within each task. We will describe the functions of each task and their relations using the following methodology: for each management task, we first give an overview of its goals, and then describe the key functions involved in this task, and finally discuss some assumptions and explain the rationale behind our design.

4.1.11 Visual Stream Source Management in MyView

One visual stream source management service entity is associated with each visual stream source, and it is in charge of sending both the visual data and the metadata of the stream to remote users. Note that the data sending part is straightforward and involves no special treatment related to visual space management, so it is not the focus of our discussion below.

4.1.11.1 Setup Time Tasks

At the setup time, a human moderator (system administrator) will be needed to establish the visual stream model by the following steps:

- **Defining name of visual stream**: since all visual streams may be presented to the user for specifying their preference/scores, the stream name will be important for the user to distinguish between these visual streams.
- **Defining resolution constraints**: the minimum width and height of the stream, which also implicitly tell the aspect ratio for display. This constraint may come naturally from the type of the visual stream, or it may be specified by the system administrator before hand.

For example, for a camera video stream capturing a talking head at *320* pixels wide and *240* pixels high, the minimum resolution can be set as low as *160x120* pixels for acceptable visual quality, but not lower. On the other hand, for a PowerPoint slideshow with a lot of texts, although there is no indicator on the aspect ratio or resolution constraints, common sense can be used to specify the minimum resolution to be *640x480*.

- **Defining shot type and triggering event for each shot type**: we adopt an event-based method to detect shot boundaries in real time (transition from the current shot type to a new shot type). The types of triggering events and their detection method are all application specific: some events may be detected based on people and objects depicted in the visual stream and their movement and activities; some events may be defined based on the time how long the stream has been in the current shot type; some events may be manually detected by human operators at the source of the stream via a user interface. All that is important to MyView framework is that the shot type should be easy to understand by the users and the user's interest in a visual stream is assumed to stay the same during a shot (and it may or may not change as shot type changes). For example, for a camera video stream showing the head of a person, "talking shot" and "silent shot" will definitely be easy to understand by the user, and they should be separated as two shot types since the user will likely have different interest in them.
- **Specifying default starting shot type**: at the very beginning of the application, no triggering event has occurred yet, so a default shot type needs to be specified for each visual stream.

4.1.11.2 Runtime Tasks

For each visual stream, at runtime, the management module has the following functions:

- **Streaming visual data**: during the runtime of a multi-stream application, if the source of the visual stream is not located at the user site (such as a remote video camera), then the visual stream source management module will need to send the visual data to the remote user sites via the transportation channel. Otherwise, such as for the case of a web browsing application where the visual stream of the web content is generated right at the user site's browser interface, no data streaming is needed.
- **Reporting shot type change**: whenever the current shot type of the visual stream is changed, the management module will report the new shot type to the stream selection management module of all receiving users immediately so that each module will be able to check for potential update of the set of selected streams. For example, for a video camera showing a person's head, when that person begins to talk, this event triggers the shot type of this visual stream from "silent shot" to "talking shot", and this change in shot type will be reported to all users receiving this stream.

4.1.11.3 Discussion about Real-Time Shot Detection

Defining shot types for a visual stream and detecting real time shot type changes are the foundation of MyView framework. Since it is different from the traditional video summarization work in that the shot boundary is detected in real time rather than through offline processing, some people may find it hard to implement. We argue that this is a viable solution for the following reasons. First, real time event detection is an active research area in video content analysis, and it is already a reality in many applications, such as video

surveillance and people/object recognition. These events are already of interest to the users, and they can be utilized for guiding the stream selection through the shot based stream model. Second, even if there are some interesting events and shot types that cannot be reliably detected by computer software, human input can be used so long as the benefit provided can justify the labor cost. For example, for the case of broadcasting a soccer game from multiple cameras for millions of audiences, each camera can be assigned a dedicated human operator that labels the current shot type in real time, such as “player X (or other player)’s frontal (or profile) view with (or without) ball”, which would already be useful for audiences interesting in only seeing frontal views of players with the ball. In sum, with the growing results in automatic video understanding, more of the visual events will be detectable automatically, but our MyView framework will not be affected since it only needs to know the real time shot type change, and the underlying detection method is kept transparent to the other modules of the MyView framework.

4.1.12 Visual Stream Selection Management in MyView

The visual stream selection engine is the key component of the MyView framework. It serves as a liaison between the visual stream sources and the end user on the decision of which visual streams are selected for rendering on the screen.

4.1.12.1 Setup Time Tasks

During the setup time, the visual stream selection management module needs to collect information about all candidate visual streams and the user interest. Specifically, the following tasks are carried out:

- **Collecting shot types of each stream**: for each visual stream, the visual stream selection engine will contact its visual stream source management module for information about how many shot types that stream may expose during runtime, and the name of each shot type.
- **Collecting user preference parameters via a user interface**: the system will assign default values for the score level, minimum and maximum on screen time, transition idioms, and the user will be prompted in the user interface (application specific) to change them if he wants to. The user interface design can be application specific, and it does not need to follow any paradigm. We will give an example of the user preference user interface in Chapter 7.
- **Calculating the default stream selection at startup**: using the default starting shot type of each visual stream and the score setting for each shot type, the selection engine will be able to calculate which streams are selected for rendering at the startup of the application.

4.1.12.2 Runtime Tasks

- **Timed automata-based stream selection**: the stream selection process is modeled after a timed automaton. The state of the automaton is the current shot type combination of all visual streams, the minimum time before the next change in stream selection decision, the maximum time for changing to another stream if several streams are selected in rotation, and the flags indicating whether there has been some shot type change before the minimum time expires and whether the current selection is the result of a user specified transition idiom. The transition of the automaton may be triggered by shot type change or

by timer events. We will present the detailed algorithms for the stream selection algorithm in Chapter 5.

4.1.13 Visual Stream Rendering Management in MyView

The visual stream rendering management module makes the decision of how to place the selected visual streams onto the given screen space. It interfaces with the user directly and determines the visual quality of the final output program of the system.

4.1.13.1 Setup Time Tasks

During the setup time, the rendering management module needs to accomplish the following tasks:

- **Registering display device**: the system should automatically detect the number of display devices used and their maximum resolution. For example, on the Windows platform, multiple display devices can be used as an “extended desktop” so that they form a high resolution virtual desktop. Such information can be retrieved programmatically so that the rendering management module knows the maximum screen space for the application.
- **Specifying screen areas**: by default, MyView can assume all detected display devices are to be used for rendering the selected streams and each display device is taken as one screen area. A screen area setup tool should be provided to allow the user to manually specify the screen area(s) she wants to allocate for this application.

4.1.13.2 Runtime Tasks

- **Calculating screen layout pattern for the selected visual stream**: whenever the stream selection engine changes the set of selected streams, the rendering management entity will

recalculate how much screen space to allocate to each of these visual streams and what is their relative position within the screen areas. The specific calculation algorithm will be described in details in Chapter 6.

4.1.13.3 Discussion about Screen Layout Management

The notion of screen layout pattern is unique in our visual space management framework in that traditional solutions only assume very simple and standard screen layout assumptions, such as a single display device, a single screen area, and a few fixed screen layouts such as 2x2 tiled window or full screen. We are the first to identify the problem that in a multi-stream environment, multiple streams with different resolution constraints will likely be selected to be displayed on one or multiple screen areas as specified by the user. This is already happening for the case of multi-model distributed conferencing where multiple communication channels have their own display windows that are shown on screen together. Even for the case of interactive TV, this is also the future trend if we consider the output resolution will become higher (especially for the case of High Definition TV) and the content will diversify from one simple TV program to several other information sources (weather information, stock price, information about related character/products for a specific show, etc.).

4.2 Example System Using MyView Framework

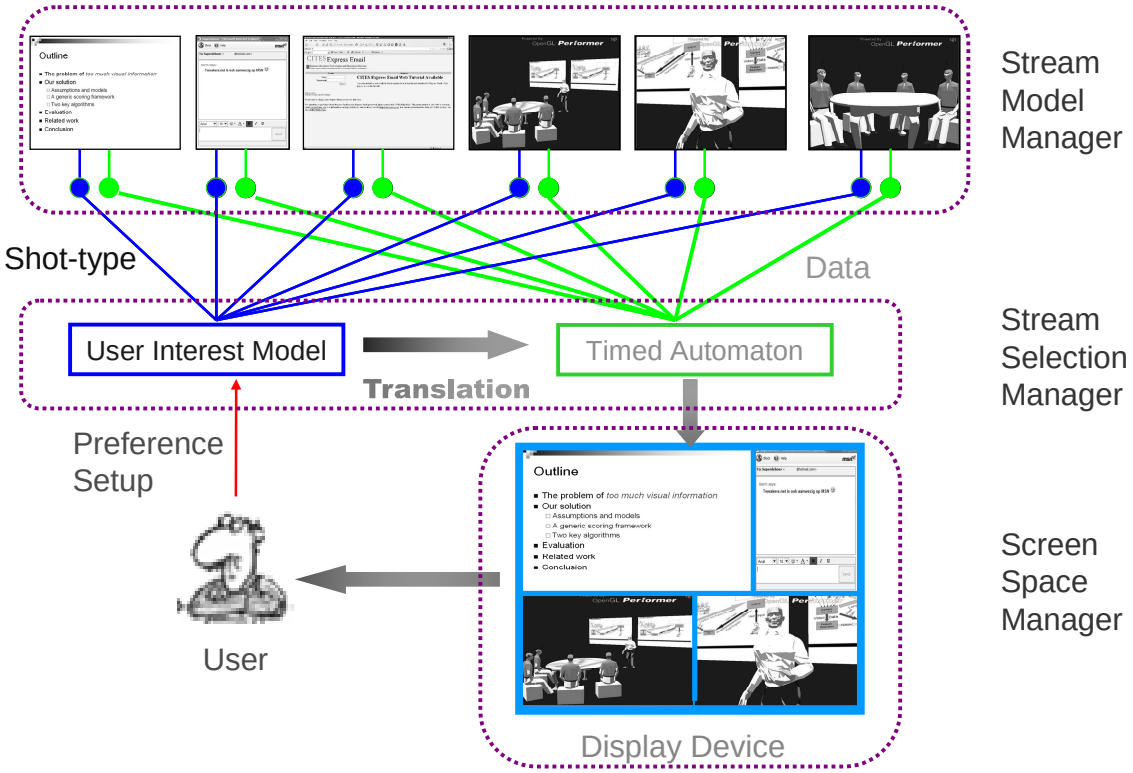

Figure 12. Overall Architecture of MyView

Figure 12 illustrates the overall architecture of an example distributed class system that follows the MyView framework. A stream source manager is assigned to each visual streams source to maintain the current shot type and other properties of that stream. The user interest model collects all the possible shot types from all the visual streams (the blue lines) and the user's preference setting (the red line) to build the user interest model, which is then translated into a timed automaton. The timed automaton is used to determine which visual streams are selected in runtime based on both the shot type changes (the green lines) and other events. Finally, the screen space manager arranges the selected streams on the screen areas specified by the user

according to the resolution constraints of these streams. In the next 3 sections, we will describe the details of the system design and how the three management tasks in MyView framework are carried out by the three management entities (stream model manager, stream selection manager and screen space manager).

4.2.14 Visual Stream Model Manager

For the visual stream model manager, a key application specific aspect is how to define shot type for the visual streams of conferencing channels. Therefore, as an example, we enumerate all the shot type definitions related in a typical distributed class application. Note that the shot type definition can be very flexible, and it may be completely different for other system designers or other application scenarios.

- PowerPoint Slideshow Streams
  - Emphasis Shot: whenever a new slide or a new animation is presented, we define the PowerPoint slideshow stream to be in the “new slide shot”.
  - Ordinary Shot: whenever the slideshow stays the same (no animation or no new slide) after a specific time period (*K* seconds), the stream turns to “stale slide shot”, meaning that the information shown is stale. The threshold *K* seconds has a default value of 120 seconds, but it can be specified by the users any time via the user interface.
- Video Camera Streams
  - Talking Shot: when the person shown in the video is talking, then this stream is defined to be in the “talking shot”
  - Silent Shot: when the person shown in the video is not talking, we define this stream to be in the “silent shot”

- Whiteboard Streams
  - New Drawing Shot: whenever there is some change in the content on the whiteboard, such as adding or erasing of a stroke, the stream is in the “new drawing shot”.
  - Stale Drawing Shot: whenever the slideshow stays the same (no adding or erasing of strokes) after a specific time period (*K* seconds), the stream turns to “stale drawing shot”. Again, the threshold *K* seconds has a default value of 120 seconds, but it can be specified by the users any time via the user interface.
- WWW Co-browsing Streams
  - New Webpage Shot: whenever a new URL is browsed to, this stream is in the “new webpage shot”.
  - Stale Webpage Shot: whenever the browser stays at the current web page for over *K* seconds, the stream turns into the “stale webpage shot”.
- Text Chat Stream
  - New Conversation Shot: whenever there is new text input at the chat room, this stream turns to the “new conversation shot”.
  - No Conversation Shot: whenever there has been a time period of over *K* seconds that no one is inputting any text, the text chat stream turns into “no conversation shot”.

4.2.15 Visual Stream Selection Manager

For the visual stream selection manager, the key application specific issue is the user interest learning process. Currently MyView relies on direct user input on an innovative user interface.

An example user interface for setting up preference among the visual stream's shot types is illustrated in Figure 13.

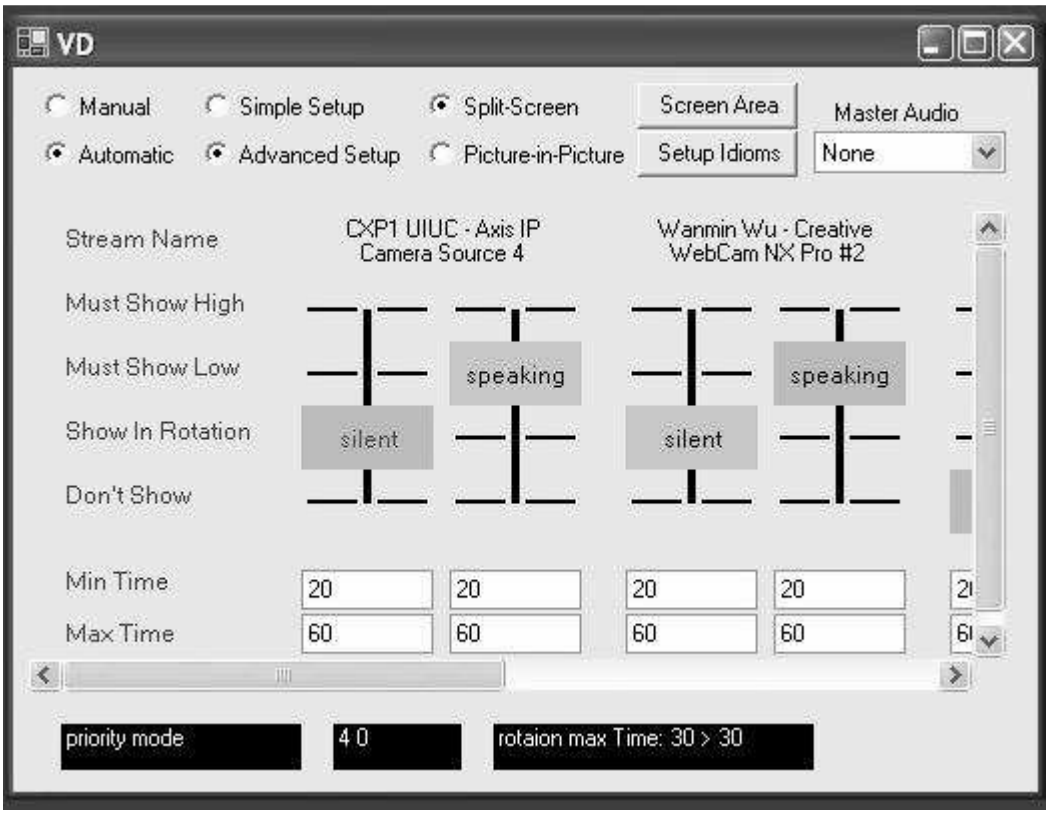

Figure 13. Screen Shot of Preference Configuration User Interface

A vertical score bar is presented on the UI for the user to tune the shot score level. It models the controls used by DJ's in real life, so it is very easy for the user to understand that the vertical track represents the range of score value and the height of the score button (with the shot type's name on it) represent the current score assignment. So if one shot type's score button is higher than another, it means the higher one is assumed to be of more interest to the user by the MyView system. As discussed earlier, the score level range is simplified to four levels, and the user can also specify the minimum and maximum on screen time using this interface.

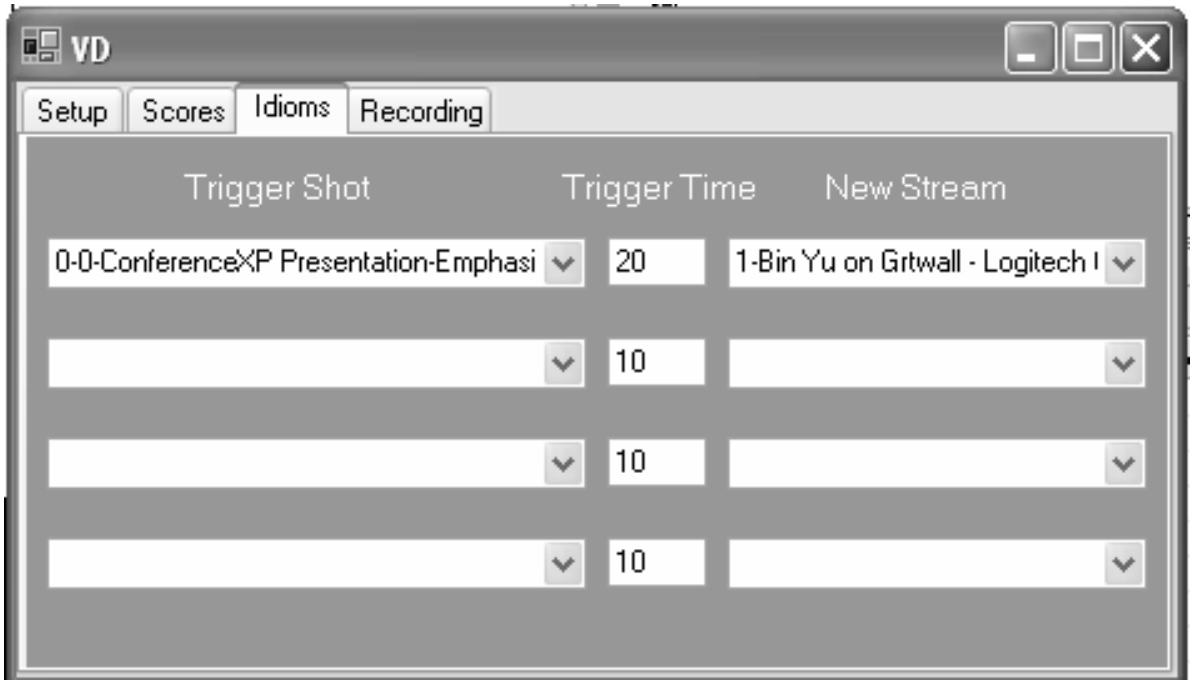

Figure 14. User Interface for Specifying idioms in MyView

The interface shown in Figure 14 allows user to specify transition idioms in the following format: *if the shot X of stream* VS_i (“Trigger Shot” in the user interface) *has been on screen for m seconds* (“Trigger Time” in the user interface), *then switch to show another visual stream* VS_j (“New Stream” in the user interface). For example, the idiom given in Figure 14 says *if the current selected shot is stream 0* (“ConferenceXP Presentation”)*’s shot type 0* (“Emphasis Shot”) *and if this keeps for at least 20 seconds* (no any other transition is fired), *then switch to display the stream 1* (“Bin Yu on GrtWall – Logitech Camera”). Note that since the stream selection manager cannot send command back to the stream source manager, so it cannot control the shot type of each stream, and that is why the “New Stream” cannot be replaced by “New Stream’s New Shot Type” because we have to switch to that new stream no matter what is its current shot type. With the scores representing the viewer preference, intelligent stream selection can be made by the computer system, which is described in details in Chapter 5.

4.2.16 Screen Space Manager

The only application specific aspect of the screen space manager is the design of user interface for specifying screen areas. In MyView, we invented a very simple method for this purpose. We allow the user to specify the number of screen areas he would like to assign for MyView, and for each screen area, a separate dialog box is shown for the user to specify its boundary.

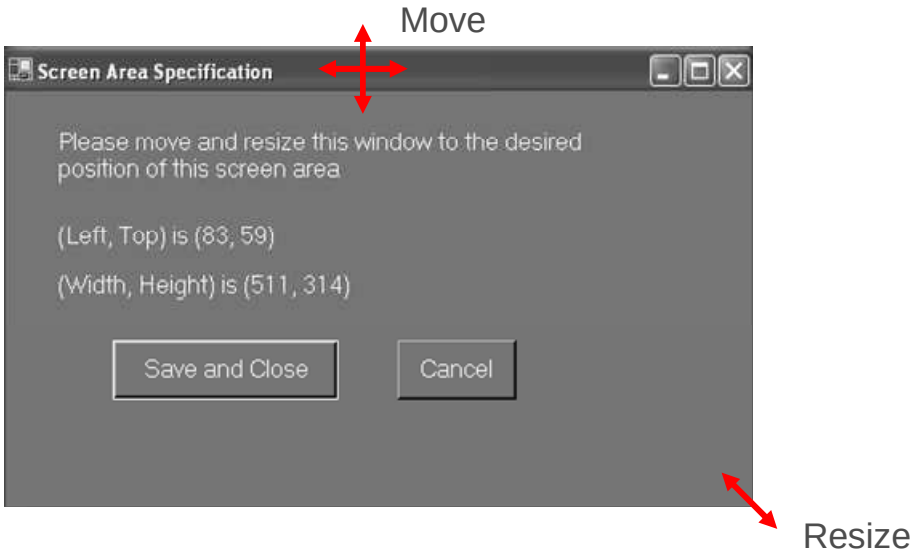

Figure 15. User Interface for Specifying Screen Area's Position and Size

Figure 15 shows the dialog for specifying screen area. The idea is that the dialog's position and size on screen represents the screen area to be used for rendering visual streams. The dialog pops up at a default size and position on the screen, and the use can *move* or *resize* it to the desired position on the screen. Once the screen areas are specified, a screen layout plan can be calculated by the computer system, which is described in details in Chapter 6.

4.3 Summary

In this chapter, we have presented the MyView framework for visual space management and described an example of how this framework can be applied to the distributed class

applications. Three main management tasks are discussed for both setup time preparation and runtime execution. We left two of the key algorithms for the next two chapters because they are very important contributions in this thesis and require a detailed discussion. In Chapter 5, we will explain the algorithm for automatically generating the timed automaton that makes the decision of stream selection during runtime, and Chapter 6 presents the algorithm to calculate the screen layout for rendering the selected visual streams on the available screen space.

Chapter 5. TIMED AUTOMATON BASED STREAM SELECTION

5.1 Overview

The visual stream selection management is at the core of our solution framework, whose task is to maintain the state of the visual stream selection process according to a timed automaton. The key challenges in this step are:

- **User Interest**

The timed automaton must reflect the user interest, which includes the preference among the visual streams (reflected by the score assignment), the minimum and maximum on screen time and the transition idioms. This means that for the same set of visual streams, different users may have a completely different stream selection result, and a predefined fixed automaton for all users will not satisfy the needs. The user interest may change in real time as the application session evolves, which means the underlying timed automaton for stream selection should also be flexible to reflect such real time changes.

- **Functional Correctness**

We need to be able to verify the functional correctness of the timed-automaton-based stream selection process, especially the *liveness property* (whether the automaton will yield deadlock after certain transition path has been taken) and the *reachability property* (whether all legitimate states of the automaton can be reached in limited number of transitions).

- **State Explosion**

Since the state of the stream selection process depend on many factors, such as the number of visual streams *N*, the average number of shot types *M*, and other parameters such as transition idioms, minimum and maximum on screen time. A simple estimation tells us the number of states will be in the order of M^N and the number of transitions in the order of $N*M^N$ (a detailed complexity analysis is given in Section 5.3.3). This poses a challenge on how we can maintain the timed automaton in real time to make stream selection.

Our solution deals with these challenges by decoupling the run time execution with off line verification. Specifically, during the run time of the stream selection process, we do not generate the whole timed automaton, but only maintain the current state of the timed automaton and the list of potential transitions. When one transition is triggered, we update the state information (including the new stream selection decision). On the other hand, we have developed an automatic automaton generation algorithm that generates a full timed automaton from the user interest model conforming to the UPPAAL formal specification, which allows us to understand the complexity of the problem and verify the functional correctness of our stream selection process. With the help of the UPPAAL verification toolbox, we are able to explore all the potential states and transitions for the stream selection process.

In the remaining part of this chapter, we will first describe the algorithm for run time execution of the timed automaton for visual stream selection in Section 5.2, and then present the automatic timed automaton generation algorithm for offline formal verification in Section 5.3.

5.2 Run Time Stream Selection Algorithm

In this section, we describe the algorithm for generating the timed automaton for stream selection. We will first present an overview of the run time execution protocol, and then go into details on the state representation and the algorithm for updating the state as triggered by run time events.

5.2.17 Run Time Execution Protocol Overview

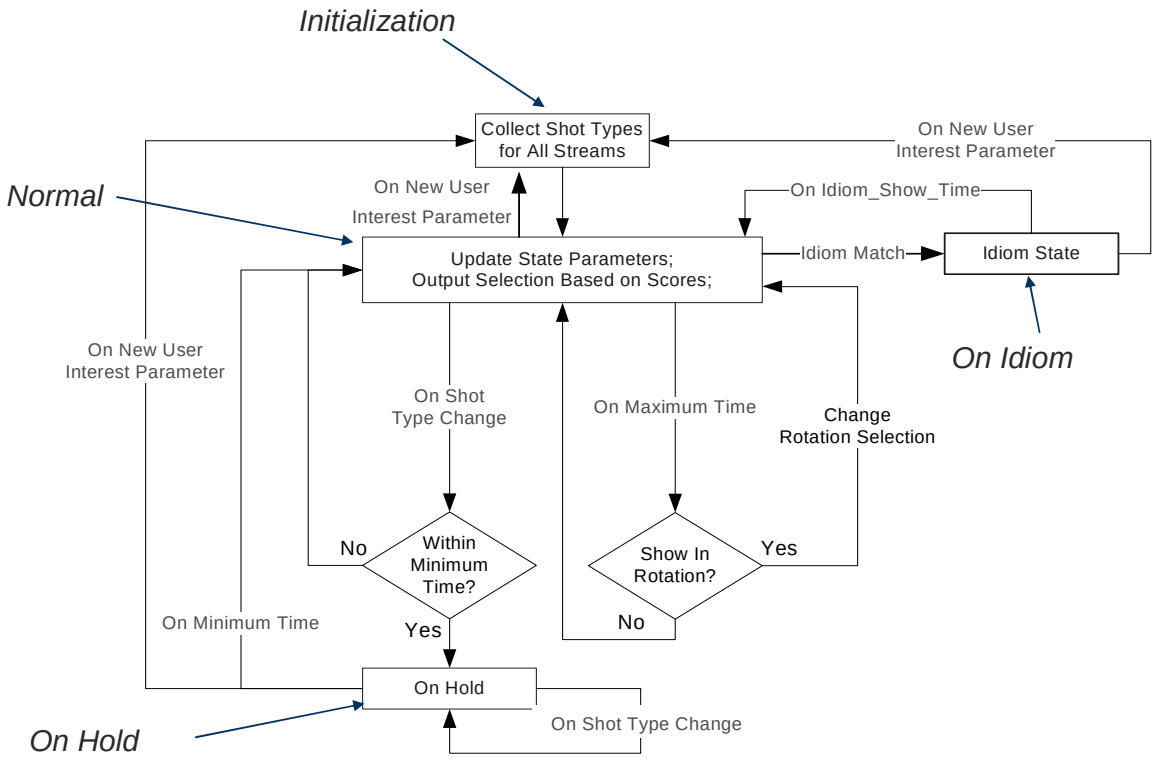

Figure 16. Runtime Algorithm for Timed Automaton Based Stream Selection

Figure 16 shows the flow chart of the runtime stream selection process.

- **Initialization**

  The state is initialized by collecting the current shot type of all visual streams and the user interest setting (scores, min/max on screen time and idioms). Whenever there is a change

in user interest setting, such as when the user changes the score of a specific shot type of one of the visual streams, the state machine will come back to the initialization state.

- **Normal Score Based Stream Selection Mode**

  During the normal execution of the stream selection engine, the system will make the stream selection decision based on the current scores of all the visual streams' current shot type. Specifically, if there are any streams with score 2 or 3, they will be selected; otherwise, if there are one or multiple streams with score 1, then they are going to be selected one by one in rotation; finally, if all streams' current shot types are rated with score 0 by the user, then the default stream VS_0 will be selected for output. Once the selection is determined, the other state parameters are also updated, such as the minimum on screen time, and maximum on screen time if a stream rotation is in action, and what are the potential matching idioms.

- **Shot type change → On Hold**

  Whenever there is a shot type change of any of the visual streams, if the change happens after the minimum on screen time of the current state has passed, then the timed automaton will stay in the normal execution mode and update selection accordingly. Otherwise, the timed automaton will enter the "On Hold" mode. In the "On Hold" mode, the stream selection cannot be changed because the minimum on screen time of the currently selected streams has not been passed. Therefore, in the "On Hold" mode, all shot type changes will be absorbed with no action. When the minimum on screen time has passed, the stream selection engine will go back to the normal score based selection mode and update state and selection accordingly.

- **Maximum on screen time**

  If multiple streams are scored 1 and are being selected in rotation, then whenever the maximum on screen time expires, the stream selection engine will rotate to the next stream with score 1 and a higher index.

- **On Idiom Matching**

  When the current state of the timed automaton matches the condition of a user specified idiom, (such as the case VS0 in its shot type A has been selected as output for exactly 30 seconds), the stream selection will enter the “On Idiom” mode. In this mode, all the shot type changes are ignored. When the specified on show time for the idiom has been satisfied, the stream selection restores to the normal execution mode again and update selection accordingly.

5.2.18 State Representation and Event Handling

5.2.18.1 State Representation

The state of the stream selection process is parameterized as follows:.

- **Selection_Time:** how much time has passed since the last time the selection is changed.
- **Shot_Type_Combination:** This is the current shot type of every visual stream, indicating that change of shot type of any of the visual streams will lead to a change of state of the stream selection manager.
- **Selected_Streams:** This is the set of visual streams selected for rendering. This is included in the state representation because for the same combination of shot types of all the visual streams, the selected visual streams may be different in difference

circumstances. By default, it should be based on the score levels of each visual stream's current shot type. That is, the selected stream index will be all visual streams in a shot type with score of 2 ("must show with low priority") or 3 ("must show with high priority"). When there are no such streams, then there will be only one selected visual stream that rotates through all the visual streams with scores of 1 ("show in rotation"). In the rare case that all visual streams are in a shot type with score of 0 ("do not show"), then by default the VS_0 will be selected. Another special case is caused by user specified idioms. An idiom may require the automaton to change the selected visual streams even when no score change event occurs.

- **Min_Time**: Before this minimum time expires, the selected visual streams should remain the same. The minimum time for a state is then determined by the maximum of all the minimum time specified by the user for all the selected visual streams.
- **Max_Time:** When the selected visual streams are shown in rotation, if this maximum time expires before any shot type change event fires, the selection rotates to the next visual stream. Since the maximum time is only valid when the stream selection manager is selecting one of the visual streams with score of "show in rotation", the maximum time for a state is determined by the maximum time of the currently selected visual stream.
- **On_Hold:** this is a Boolean flag indicating whether a change of output is pending (on hold) because of the minimum time limitation. For example, if a shot type change occurs before the minimum on screen time of the currently selected streams, then we update the state for the new shot type combination of all visual streams without updating the output.

We also mark the state to be On_Hold so that when the minimum time has expired, we will update the output stream selection accordingly.

- **In_Rotation:** this is a Boolean flag indicating whether we are showing a set of streams in rotation. This situation occurs only when multiple visual streams are in a shot type with score 1 and other visual streams are all in a shot type with score 0.
- **In_Idiom:** this is a Boolean flag indicating whether we are showing a set of visual streams based on a user specified idiom.

5.2.18.2 Event Handling Routines

The state of the stream selection engine is updated when the run time events occur. To help the reader better understand the state transition process, we present detailed algorithm of the event handling routine in pseudo code with detailed explanations.

1. Common Stream Selection Routine

   This is a shared routine used by multiple event handlers below to calculate which visual streams should be selected for output. It selects the top ranking stream(s) based on the current score of each stream.

```
Compute_Selected_Streams()
    High_Score_Stream_Set  = Set of visual streams with shot types scoring 2 or 3;
    If ( High_Score_Stream_Set is not empty ) {
            // if there are streams with high priority scores, then they are selected for output
            Selected_Streams = High_Score_Stream_Set;
            Min_Time = Maximum of minimum on screen time for selected streams' current shot types
            Max_Time = -1; // not used
            In_Rotation = false;
    } Else {
            Rotation_Stream_Set  = Set of visual streams with shot types scoring 1;
            If ( Rotation_Stream_Set is not empty ) {
                    // if one or more streams are of rotation score, they are selected for viewing in rotation
                    Selected_Streams = Rotation_Stream_Set;
                    In_Rotation = true;
                    Max_Time = maximum on screen time of the selected stream's current shot type;
                    Min_Time = minimum on screen time of the selected streams' current shot type;
            } Else {
                    // the rare case that all visual streams are in shot types scoring 0, so select stream 0
                    Selected_Streams = {0};
                    In_Rotation = false;
                    Min_Time = minimum on screen time of visual stream 0;
                    Max_Time = -1; // not used
            }
    }
```

2. Shot-type Change Handling Procedure

This routine handles the event that a stream's shot-type is changed. It makes the decision of switching to *on_hold* state based on whether minimum on screen time has passed.

```
    On_Shot_Type_Change_Event(Stream_Index, New_Shot_Type)
    Shot_Type_Combination[Stream_Index] = New_Shot_Type;
    If ( Selection_Time > Minimum_Time ) {
        // we have passed the threshold to safely update selection without causing too frequent screen changes
        Compute_Selected_Streams();
        On_Hold = false;
    } else {
        // do not update output, but mark the flag that we are on hold for a change of output
        On_Hold = true;
    }
```

3. Maximum On Screen Timer Expiration Event Handler

This routine handles the timer event that signals we have passed the maximum time for screen updates. It switches to the next stream on the rotation list or simply restart the selection process otherwise.

```
On_Max_Timer_Event()

If ( In_Rotation ) {
    // if we are in the state of showing multiple streams in rotation, rotate to the next stream with score 1
    New_Selected_Stream = Randomly Choose One Stream with Score 1;
    Selected_Streams = New_Selected_Stream;
    Max_Time = maximum on screen time of the New_Selected_Stream's current shot type;
    Min_Time = minimum on screen time of the New_Selected_Stream's current shot type;
} else {
    // restore the normal output stream by computing the selection from scratch
    Compute_Selected_Streams();
}
```

4. Handling Procedure for Matching Idioms

This routine handles the event that an idiom has been matched by setting up the maximum and minimum on screen time timers.

```
On_Matching_Idiom(Idiom_Show_Time, Idiom_Target_Stream)
In_Idiom = True;
// switch to show the streams specified in the idiom.
Selected_Streams = Idiom_Target_Stream
// setup timer so that we switch out of the on_idiom state once we pass the time specified by the idiom
Max_Time = Idiom_Show_Time;
Min_Time = minimum on screen time of the Idiom_Target_Stream's current shot type;
```

5. Handling Procedure for Minimum On Screen Timer

This routine handles the event that the minimum on screen time has passed, and resume to normal state by starting a new selection process

```
On_Min_Timer_Event
If ( On_Hold ) {
    // if we are holding off screen updates due to the minimum on screen time threshold, compute selection.
    On_Hold = false;
    Compute_Selected_Streams();
}
```

5.3 Automatic Timed Automaton Generation Algorithm

In this Section, we present the algorithm to generate the whole stream selection timed automaton. The generated automaton will not be used for on line execution of stream selection because of its complexity. Instead, it is executed offline to generate a full timed automaton including all the possible states and transitions that may or may not be executed during a runtime session. Such an automaton is used to verify the functional correctness of the timed automaton using the UPPAAL toolbox, as will be described in Chapter 8.

To keep the discussion concise, we will focus on the knowledge representation (in consistence with UPPAAL syntax) and present intuitive explanations of the algorithm through several working examples with increased complexity. The complete pseudo code is given in details in Appendix A.

5.3.19 Definitions, Notations and Parameter Representations

5.3.19.1 Global Definitions

Global definitions include the following variables (readers may refer to the Section 3.2.8 for introduction on the description language for timed automaton conforming to the UPPAAL standard):

- ***N***: number of visual streams. These *N* streams will be assigned a stream index from *0* to *N-1*, so they are represented as VS_1 to VS_{N-1}.
- M_i: number of shot types for visual stream VS_i, and each shot type will be assigned a shot type index from *0* to M_i*-1*.
- Channels: for each visual stream VS_i with M_i shot types, M_i *channels* are defined to be "*c[stream_index]_[shottype_index]*". Each channel represents the shot type change event for that shot type and is used to trigger the transitions of the stream selection manager. For example, "*c0_2*" is the channel that represents the event that stream VS_0 is changing to its shot type *2*.
- *Min[i][j]*: the minimum on screen time for VS_i's shot type *j* (user specified value or default system value)
- *Max[i][j]*: the maximum on screen time for VS_i's shot type *j* (user specified value or default system value)
- *S[i][j]*: the score for VS_i's shot type *j* (user specified value or default system value)

- *curtime*: the clock for the time how long the current selection has been made. This clock will be reset to zero every time a new selection is made so that it always represents how long the automaton has stayed with the current selection.
- *selected[i]*: whether visual stream *i* is selected for output or not.

5.3.19.2 State Representation

The state of the stream selection manager is determined by the following parameters

- **State Index**: this is the unique index of each state.
- **Shot Type Combination**: This is the current shot type of every visual stream, indicating that change of shot type of any of the visual streams will lead to a change of state of the stream selection manager. Formally, suppose there are totally *N* visual streams, and they are represented as VS_0, $VS_1 \ldots VS_{N-1}$. Each visual stream VS_i may have *Mi* shot type choices, represented as Vi_1, $Vi_2 \ldots Vi_{Mi}$. Therefore, the shot type representation for a particular state representation is a string of *N* digits, with the i^{th} digit representing the current shot type of VS_i.

  For example, suppose there are only two visual streams VS_0 and VS_1 involved in a CVS rendering session. VS_0 has three shot types (“0”, “1” and “2”) and VS_1 has four shot types (“0”, “1”, “2” and “3”). Then the shot type representation portion of any state representation will have one of the following 12 values: “00”, “01”, “02”, “03”, “10”, “11”, “12”, “13”, “20”, “21”, “22”, and “23”.
- **Selected Stream Index**: This is the set of visual streams selected for rendering. This is included in the state representation because for the same combination of shot types of all the visual streams, the selected visual streams may be different in difference circumstances.

By default, it should be based on the score levels of each visual stream's current shot type. That is, the selected stream index will be all visual streams in a shot type with score of 2 ("must show with low priority") or 3 ("must show with high priority"). When there are no such streams, then there will be only one selected visual stream that rotates through all the visual streams with scores of 1 ("show in rotation"). In the rare case that all visual streams are in a shot type with score of 0 ("do not show"), then by default the VS_0 will be selected. Another special case is caused by user specified idioms. An idiom may require the automaton to change the selected visual streams even when no score change event occurs. Formally, the selected stream index is a string of digits of the indexes of all the selected streams. For example, "01" means VS_0 and VS_1 are selected for rendering.

- **Minimum Time**: Before this minimum time expires, the selected visual streams should remain the same. The minimum time for a state is then determined by the maximum of all the minimum time specified by the user for all the selected visual streams. For example, suppose there are only two visual streams VS_0 and VS_1 involved in a CVS rendering session. VS_0 has three shot types and VS_1 has four. Then if VS_0 and VS_1 are both selected for rendering, and VS_0 is in shot type 2 and VS_1 is in shot type 3, then *Minimum Time* = *Max { Min[0][2], Min[1][3] }*.
- **Maximum Time**: When the selected visual streams are shown in rotation, if this maximum time expires before any shot type change event fires, the selection rotates to the next visual stream. Since the maximum time is only valid when the stream selection manager is selecting one of the visual streams with score of "show in rotation", so the maximum time for a state is determined by the maximum time of the currently selected visual stream. Therefore, $Maximum\ Time = MaxT_{selected\ stream\ index}$.For example, suppose

there are only two visual streams VS_0 and VS_1 involved in a CVS rendering session. VS_0 has three shot types and VS_1 has four. Suppose VS_0 is in shot type 2 and VS_1 is in shot type 3, then the stream selection manager will select VS_0 first, and when VS_0's shot type 2's maximum time expires, it will turn to select VS_1.

5.3.19.3 Transition Representation

The transitions of the automaton are represented by the following parameters:

- **From state index**: at which state is this transition considered
- **To state index**: the index of the next state this transition leads to
- **Timing constraint**: in the form of "curtime comparison_sign TIME_THRESHOLD". comparison_sign may be the following values: "<", "≤", "==", "≥", and ">". This constraint is to express the minimum or maximum time for this transition to be enabled.
- **Triggering channel**: the channel representing the change of shot type that leads to this transition to be fired.
- **Update**: some transition may update the value of global variables, such as resetting the clock or changing the selection list.

5.3.20 Automata Generation Algorithm

To model the real time dynamics of a visual stream selection system, a set of automata need to be created: *N* stream automata representing *N* visual stream, and one stream selector automaton representing the core stream selection decision engine. Each visual stream automaton's state represents the current shot type of that stream, and the transitions represent the change of shot type in runtime. Each visual stream automaton is linked to the stream selector automaton via

"channels", so that whenever there is a shot change of a visual stream, it will trigger a state change of the stream selector automaton. The stream selector automaton is also affected by a real time clock representing how much time has passed since the last state change. The output of the stream selector automaton (changed as the result of its state transition) is the decision of currently selected visual streams.

The construction of the visual stream automaton is fairly straightforward. For each visual stream, we need to create one state for each of its shot types, and then create two transitions in opposite directions between each pair of shot types that may change between each other.

The construction of the stream selector automaton is more complicated. We will describe the automaton generation algorithm in three steps with increasing complexity. We will first describe how to generate a simple automaton that only includes shot type change triggered transitions and the maximum timer triggered selection rotation. Second, we add more states and transitions to this automaton to support the minimum timer constraint. Finally, we add more states and transitions to support the idioms specified by users.

5.3.20.1 Basic Automaton with Support for Shot Type Change and Maximum Timer Events

We use an example automaton to illustrate the automaton generation process, and the pseudo code is given in Appendix A.1. Let us assume there are two streams (stream 0 and stream 1), and each stream will have two shot types (shot type 0 and shot type 1). Shot type 0 of both stream is assigned the score level 0, and shot type 1 of both stream is assigned score level 1. Then after the generation of states and transitions based on the algorithm described above, we will get an automaton as shown in Figure 17

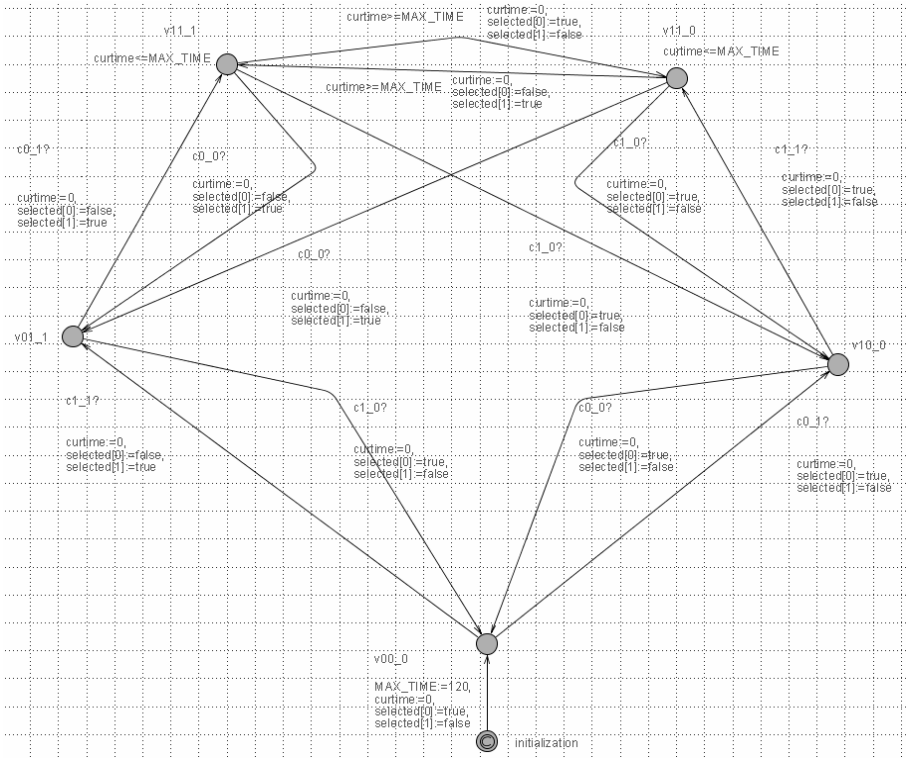

Figure 17. Basic Automata with Transitions Triggered by View Change and Maximum Timer

There are totally 5 states besides the initialization state at the bottom:

- "*V00_0*" represent the state that stream *0* is in shot type *0* (the first *0*), stream *1* is in shot type *0* (the second *0*), and stream *0* is selected (the third *0*).
- "*V01_1*" represent the state that stream *0* is in shot type *0* (the *0*), stream *1* is in shot type *1* (the first *1*), and stream *1* is selected (the second *1*).
- "*V10_1*" represent the state that stream *0* is in shot type *1* (the first *1*), stream *1* is in shot type *0* (the *0*), and stream *0* is selected (the second *1*).
- "*V11_0*" represent the state that stream *0* is in shot type *1* (the first 1), stream *1* is in shot type *1* (the second *1*), and stream *0* is selected (the *0*).

- "*V11_1*" represent the state that stream *0* is in shot type *1* (the first *1*), stream *1* is in shot type *1* (the second *1*), and stream *1* is selected (the third *1*).

8 of the 10 transitions between the 5 states in the automaton represent shot type change triggered state changes, and they are all marked with the trigger in the form of "*c[stream_index]_[shottype_index]?*". For example, the transition from *V00_0* to *V01_1* is marked with the event trigger channel "*c1_1?*", meaning that the change of shot type of stream *1* (the first *1*) to its shot type *1* (the second *1*) will trigger the state change from *V00_0* to *V01_1*. The other 2 transitions between *V11_1* and *V11_0* represent the case that the two streams are selected in rotation, and these transitions are enabled by the maximum timer. This is enforced by two conditions. First, the invariant of the state *V11_1* and *V11_0* says "curtime $\leq$ MAX_TIME", meaning that the automaton must transit out of these states within MAX_TIME units. On the other hand, the 2 transitions between them says "curtime $\geqslant$ MAX_TIME", meaning that these transitions are enabled only after the automaton has stated in one of the two states for at least MAX_TIME units. Therefore, their combined effect is that if no shot type change occurs, the transition will be fired exactly at MAX_TIME.

5.3.20.2 Automaton with Support for Minimum Time Constraint

The automaton in the last example does not support the minimum on screen time constraint, so it will lead to immediate screen update whenever there is a triggering event (such as a shot type change) even if there has just been a screen update. In this section, we describe the enhancement that add the support for the minimum on screen time constraint. The pseudo code can be found in Appendix A.2.

We will continue on the example system we presented in the last Section. After adding states and transitions to support minimum time constraint, the new automaton is shown in Figure 18 (overview) and Figure 19 (zoom in view).

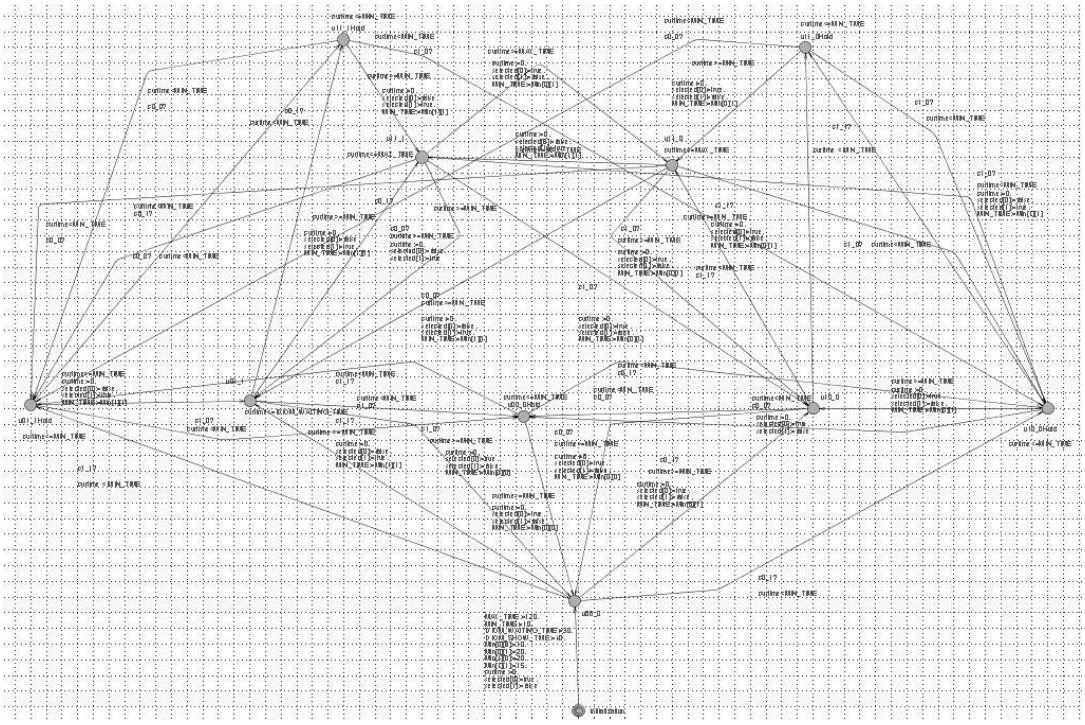

Figure 18. Example Automata With Support For Minimum Time Constraint

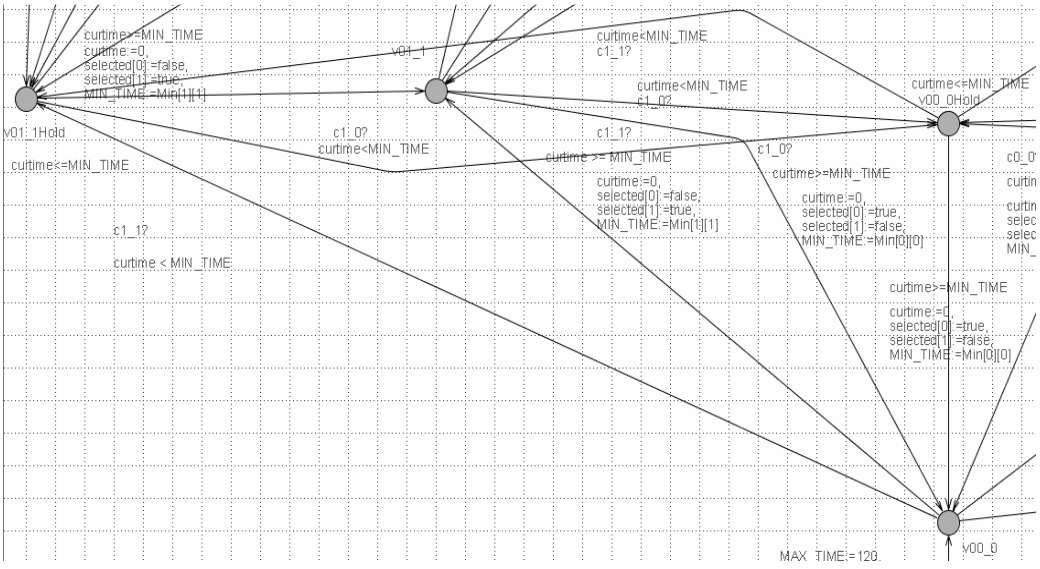

Figure 19. Details of Portion of the Enhanced Automata Shown In Figure 17

In the new automaton, the following changes are made:

- 5 hold states are added corresponding to the 5 original output states: *V00_0Hold, V01_1Hold, V10_0Hold, V11_0Hold, V11_1Hold*. From each of these hold states, one transition is added to the corresponding output state. For example, the transition from *V01_1Hold* to *V01_1* is marked with the condition "*curtime* $\geqslant$ *MIN_TIME*", meaning that if the automaton is in the hold state and more than MIN_TIME has passed since the last time selection was changed, then the automaton should change to the output state *V01_1*. Note that such transitions will also reset the clock "*curtime := 0*" and MIN_TIME.
- All the original shot type-based transitions are enhanced with a new condition "*curtime* $\geqslant$ *MIN_TIME*", meaning that these transitions will not be enabled when the clock has not passed MIN_TIME.
- New transitions are added from original output states to hold states to represent the case when a shot type change occurs before the MIN_TIME duration is passed. For example, the transition from state *V00_0* to *V01_1Hold* has a guard condition "*curtime* < *MIN_TIME*", meaning that if the curtime clock is less than MIN_TIME, this transition will be fired and the automaton will change to the *V01_1Hold* state.
- New transitions are added between the Hold states to represent the case when a new shot type change event occurs while the automaton is already in a Hold state.

5.3.20.3 Automaton with Support for Idiom Based Transitions

Each idiom specifies that if the current selected streams are in the *Idiom_Current_Selection_List* with a specific *Idiom_Shottype_Combinataion* for *Idiom_Waiting_Time,* then change selected streams to *Idiom_New_Selection_List* for

Idiom_Show_Time. To support such idioms, we need to add the states that represent the new selection state, and also transitions to and from them. The pseudo code can be found in Appendix A.3

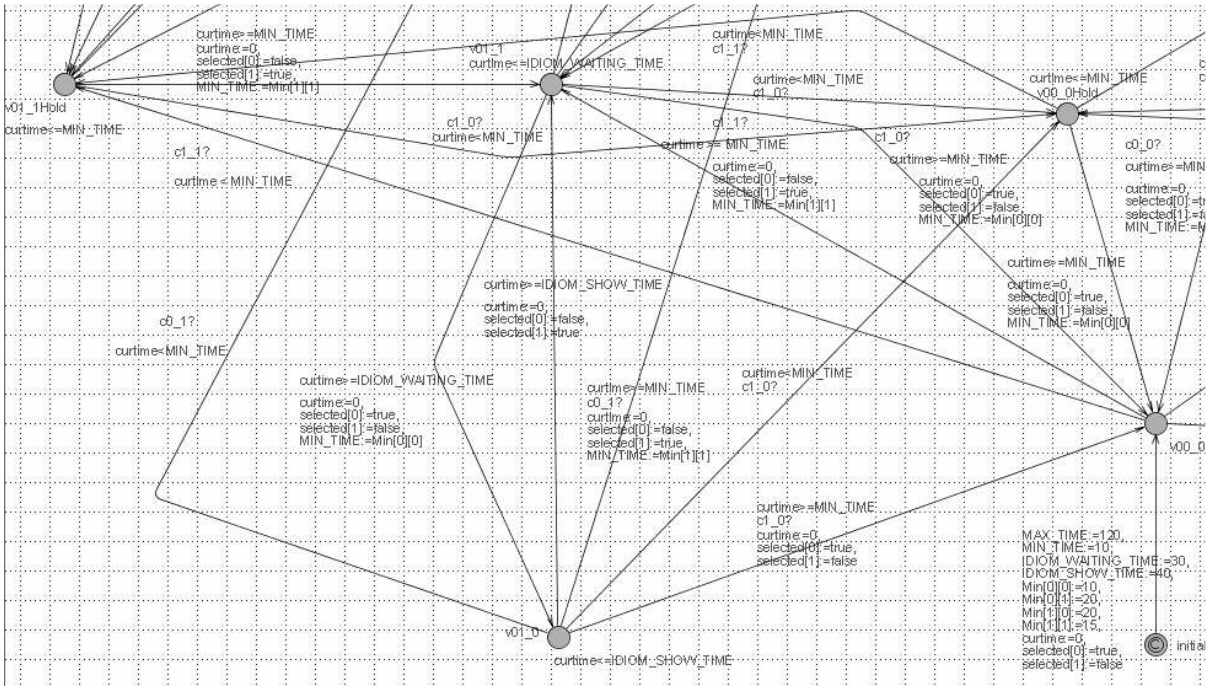

Figure 20. Illustration of Changes Added To Support Idioms

In this example, we added the support for the following idiom: if the shot type combination remains *01* (stream *0* is in shot type *0* and stream *1* is in shot type *1*) for longer than *IDIOM_WAITING_TIME*, then switch to select stream *0* for *IDIOM_SHOWING_TIME*. The changes to the automaton are illustrated in Figure 20.

The major changes are the following:

- There is a new state added *V01_0*, and there is a new transition from the state *V01_1* to *V01_0*. The idiom is enforced by the following two conditions: the state *V01_1* is enhanced with a new invariant "*curtime* ≤ *IDIOM_WAITING_TIME*", and the transition

from *V01_1* to *V01_0* has a condition "*curtime* ≤ *IDIOM_WAITING_TIME*". Their combined effect is that the transition will take place exactly after the automaton has stayed in the *V01_1* state for *IDIOM_WAITING_TIME* and not any shot type change has occurred. There is another new transition from the state *V01_0* back to the state *V01_1*, and it is guaranteed to occur after the automaton has stayed in the state *V01_0* for *IDIOM_SHOW_TIME* if no shot type change event occurs.

- There are 5 outgoing transitions from the new state *V01_0* to all the states that *V01_0* is linked to. They represent the cases that while the automaton is in the idiom specified selection state, some shot type changing event occurs to move the state of the automaton to a hold state for some other output state.

5.3.21 Complexity Analysis

One way of measuring the complexity of the stream selection process is to calculate the number of states and transitions in the timed automaton. For the simplicity of analysis, we assume there are *N* visual streams and *M* shot types for each visual stream. We also ignore the user specified idioms because their low probability will not affect the order of magnitude of the complexity estimation.

- **Total number of states**

The number of combination of shot types for the *N* visual stream is M^N. If we ignore the case of rotational stream selection, then the total number of "normal" states will be M^N. Since there is one "hold" state for each normal state, the total number of states of the timed automaton will be $2 * M^N$. For example, for the case of $M = 4$ and $N = 8$, this is $2 * 4^8 = 131072$.

- **Total number of transitions**

For each normal state, if any one visual stream changes its shot type, then it will lead to a transition to another state. Therefore, there will be *N* outgoing transitions, so the total number of transitions between normal states will be $N * M^N$. For each such "normal" transition from state A to B, there will be a transition from the state A to the hold state of state B and a transition from the hold state of state A to the hold state of B. In addition, for each normal state, there will be one transition from its hold state to itself, Therefore, the total number of transitions will be $(3N+1)*M^N$.

Overall, the order of magnitude of the problem grows exponentially as the number of visual streams increase: $O(M^N)$ for number of states and $O(N*M^N)$ for number of transitions. If we consider the application of multi-view broadcasting of a soccer game, each video camera stream may have at least *4* shot types, and there may be over *20* video cameras, then the stream selection process will have to work with over *1099511627776* states and *67070209294336* transitions. The implication of such high level of complexity are:

First, for large scale wide application of visual space management, we cannot rely on manually specified automaton. Instead, automatic stream selection can save a lot of work and time, and it also allows us to formally verify the functional correctness and desired behavior of the stream selection process.

Second, even though we should rely on timed automaton based approach, it is not practical to generate the whole timed automaton in real time. Therefore, some simulative approach as proposed in Section 5.2 needs to be adopted to simulate the run time state evolution of the stream selection automaton.

5.4 Summary

In this chapter, we presented the algorithm to generate the timed automaton for stream selection at runtime, and also to automatically generate the timed automaton offline according to the UPPAAL formal specification. The simulation algorithm allows the system to work efficiently at runtime, while the automaton generation algorithm allows us to study the functional correctness and complexity of the whole stream selection process.

Chapter 6. SCREEN LAYOUT CALCULATION

The screen space of the display devices is a critical resource that calls for careful management. It is where visual information carried by one or multiple visual streams is presented to the user. In our solution, the question of "which visual streams are put on screen" is handled by the Stream Selection Management, while the questions of "given that these streams are to be shown, how to position them on the display devices' screens?" is answered by the visual stream rendering management. In this Chapter, we discuss the visual stream rendering management and the corresponding screen layout calculation algorithm to arrange the *K* selected visual streams on to the screen space.

6.1.22 Problem Statement

Given a list of *K* selected visual streams, the screen space manager needs to position them into one or multiple screen areas (specified by the user) and resize them to fit in the screen area boundaries. Intuitively, the rendering result depends on (a) the size and number of screen areas, and (b) the resolution constraints and number of selected visual streams.

Formally, the problem of screen layout management can be described as follows: Given a set of *K* selected visual streams VS_0 to VS_{K-1} in the order of decreasing score level (as learned from the user's input), and the corresponding Resolution Constraint vectors RC_0 to RC_{N-1}, and also *L* candidate screen areas SA_0 to SA_{L-1} in the order of decreasing priority specified by the user, output a ***screen layout plan*** that consists of a list of subwindows SLP_1 to SLP_N, where SLP_i represents the position of VS_i, in the form of *SLPi = <SA_index, left, top,*

width, height>, where *SA_index* specifies which screen area to use in case of $M > 1$, and left and top specify the coordinate of the left/top corner of the display window for VS_i relative to the left/top coordinate of the parent screen area, and width and height represent the new size of the display window of VS_i. The ***optimal screen layout plan*** should fulfill the following rules:

Rule 1. Maximized screen utilization: the screen space utilized by all visual streams should be maximized. Note that this does not necessarily mean the total resolution of all visual streams, because some visual stream (such as video) have a specific aspect ratio (width over height), and it may not fully utilize the screen space allocated to it. For example, as shown in Figure 21 below, if a screen region of 600 pixels in width and 150 pixels in height is allocated to a video with aspect ratio of 4 : 3, then it will only utilize 200x150 pixels out of the 600x150 total screen space.

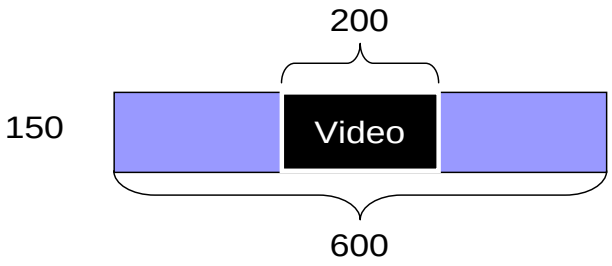

Figure 21. Illustration of Underutilization of Given Screen Area Caused by Aspect Ratio Constraint

Rule 2. Visually pleasing: the resulting arrangement of all the visual stream windows on the screen should be visually pleasing to the user in the sense that their horizontal and vertical borders should be aligned as much as possible, and their positions' relative relation should conform to the expected screen layout normally seen on TV.

Rule 3. No overlap: no two visual streams should overlap on screen. Although visual effects such as Picture-In-Picture will lead to visually pleasing video rendering, it may cause the problem of one visual stream window blocking useful information of another.

Rule 4. Resolution constraints: all the resolution constraints should be satisfied if possible. Note that the rendering algorithm may return a failure if the given screen areas are too small for the minimum resolution constraint for a visual stream.

6.1.23 Problem Analysis

The problem of screen layout management is a special case for the generic problem of two dimensional bin-packing problem, as defined below:

Given a set of *n* rectangular *items* $j \in J = \{1, \ldots, n\}$, each having *width* w_j and *height* h_j , and an unlimited number of finite identical rectangular *bins*, having width *W* and height *H*. The problem is to allocate, without overlapping, all the items to the minimum number of bins, with their edges parallel to those of the bins. It is assumed that the items have fixed orientation, i.e., they cannot be rotated. The two dimensional bin-packing problem is known to be *NP hard*.

The screen layout computation problem is even harder than the two dimensional bin packing problem because, in addition to pack the visual stream windows onto the screen areas, it has the following additional constraints:

- First, we are working with “boxes” (visual stream windows) with dynamic sizes. Each visual stream window has a resolution constraint that allows us to adapt its size based on the available space. The ratio between the rendering resolution and the minimum resolution can be different for each visual stream as well.

- Second, we are working with “bins” (screen areas) with none uniform sizes. That is, when there are multiple screen areas, some of them may be much larger (with much higher pixel resolution) than other ones. And the aspect ratio (width over height) of the screen area may also be irregular since it is freely defined by the user, so we may expect some screen areas to be like a vertical bar (width << height), while some screen areas are landscape rectangles (width > height).
- Third, the layout of the “boxes” is constrained by Rule 2, so we need to take the relative position between the visual stream windows into consideration.

Nevertheless, the good news is that the screen layout management problem is in nature a small scale problem: normally a user’s visual attention space is limited, and he would not be able to watch too many visual stream windows at the same time. Therefore, we adopt a screen layout computation algorithm that conducts an intelligent exhaustive search only over a limited solution space to find the optimal screen layout plan.

To limit the complexity of the screen layout management problem, we have made the following simplifications about the display devices. First, we only work with the display resolution (how many pixels per line and how many lines), and we do not assume the physical size of the screen. The user space is selected by the user as one or multiple screen areas, which is confined by the available physical screen space. Nevertheless, we acknowledge the fact that the same visual window of the same size (in terms of pixel width and height) will have different visual effects if displayed on a big projector screen than on a small laptop screen. Second, we do not assume the seating of the viewer with regard to the display device, but we acknowledge that the viewers looking at the same screen from the front view or side view will

have different perspectives on the same visual stream. Also, the distance between the viewer and the screen is ignored in our discussion below. This point will become more important when 3D displays are used.

6.1.24 Screen Layout Computation Algorithm

The rationale behind the algorithm is the following. First, to satisfy the resolution constraints (rule 4), we start by computing screen layout plans using the minimum width and height of each visual stream, and then resize the resulting screen layout plan to maximize the screen area utilization (rule 1). Second, when there are multiple candidate screen layout plans, we compare them using rule 1 (that is, check which plan has higher screen utilization).

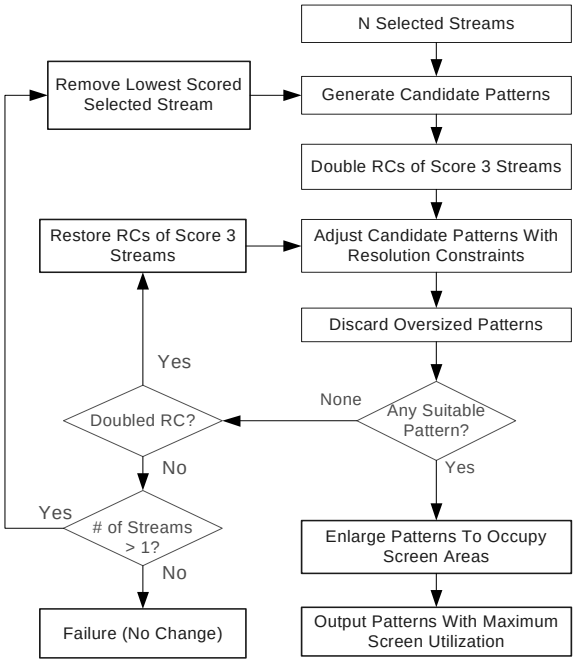

Figure 22. Screen Layout Calculation Algorithm

Figure 22 shows the flow chart of the screen layout calculation algorithm, and the complete screen layout computation algorithm is presented below.

Step 1. Generate candidate screen layout patterns based on the number of visual streams K and the number of screen areas *M*:

a) For the case of *M = 1*:

For number of rows R = 1 to K {

S = All patterns with R number of rows and in which any upper row has at least as many subwindows as any lower row;

For each pattern P in S {

Assign height of each subwindow to be screen area's height divided by number of rows;

Assign width of each subwindow to be screen area's width divided by number of subwindows on the same row;

Output P to candidate_pattern_list;

}

}

Explanation: if there is only one screen area (M=1), we generate a series of patterns with increasing number of rows from 1 (all subwindows in one row) to the number of selected streams *K* (all subwindows in one column). Then we adjust the size of each subwindow so that the whole pattern has the same size as the given screen area size.

b) For the case of *M > 1*:

T = all compositions of number of visual streams assigned for each screen area;

For each composition C in T {

For each screen area, generate a partial_pattern_list using algorithm used in the case of M = 1;

Merge all the partial_pattern_list into one candidate_pattern_list that contains combinations of all patterns of each partial_pattern_list;

}

Explanation: if there are multiple screen areas specified (M>1), we generate M patterns with the total number of subwindows in all these M patterns sums to K. since the number of subwindows in each pattern may be different, all possible combinations

T are considered. Then for each combination in T, all the M patterns are merged into a pattern list.

Example Walkthrough:

The goal of this step is to generate candidate patterns that equally divide the given screen areas into multiple rows of subwindows, which will be used as candidate patterns in the next step. For example, for the case of *K = 4* and *M = 1* (that is, four visual stream windows to be rendered on a single screen area), the following candidate patterns (shown in Figure 23) will be generated:

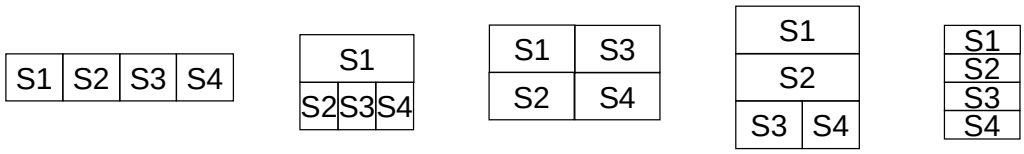

Figure 23. Example Candidate Pattern List for *K = 4* and *M = 1*

Step 2. Map the *K* visual streams with resolution constraints into the candidate patterns

For each selected visual stream with score 3, multiply its minimum width and height by 2.

***Start**:*

*Sort visual streams with the same score by the minimum resolution (minimum width * minimum height) in decreasing order*

For each pattern P in candidate_pattern_list {

For x = 1 to K {

Let SW be the xth subwindow of P;

Change SW's right boundary so that its width is at least the minimum width of the xth visual stream; if there is another subwindow to the right of SW, change its left boundary;

If (SW's height < the minimum height given in RC_x) {

Change SW's bottom boundary so that its height is at least the minimum height given in RC_x;

Change the bottom boundary of all other subwindows on the same row as SW if any;

Change the top boundary of all other subwindows on the row below SW's row if any;

}

}

If all the adjustment has been successfully done without exceeding the boundary of the given screen areas, then add this pattern P to the potential_pattern_list;

}

If (potential_pattern_list is empty) {

If (visual streams with score of 3 are using original minimum width and height) {

Return failure;

} Else {

For each selected visual stream with score of 3, divide its minimum width and height by 2;

*Go to "**start**";*

}

}

Explanation: this step resizes each candidate pattern to fit with the resolution constraints of the specified screen areas. We first enlarge the resolution specifications of those selected streams with score 3 so that they will be allocated more screen space. Then we map each of the *K* selected streams into the *K* subwindows, enlarging the subwindow when needed and shifting the other subwindows to the right and bottom of the current subwindow along the way. After we have fit all *K* streams, we check if the width or height of the resulting pattern has exceeded those of the specified screen areas, and discard a pattern if so. If we are ended with an empty list of suitable patterns, we will restore the original resolution constraints of the streams with score 3 and try again (go back to ***start***); otherwise we fail to find a suitable pattern that could fit all selected streams while obeying the sizes of the specified screen areas.

Example Walkthrough:

The goal of this step is to try to map the visual streams into each of the candidate patterns. If a specific visual stream can't be fit into a subwindow, then adjust the width or height of that subwindow using the minimum width and height of the visual stream. If this adjustment would cause the width or height of the pattern to exceed that of the given screen areas, then this pattern is ignored. For example, suppose the minimum width and height of four visual streams and the screen area are given in Figure 24 below. If we try to use the candidate pattern with four subwindows in one row, the resulting pattern will exceed the boundary of the screen area; however, if we use the 2x2 pattern, it will work well.

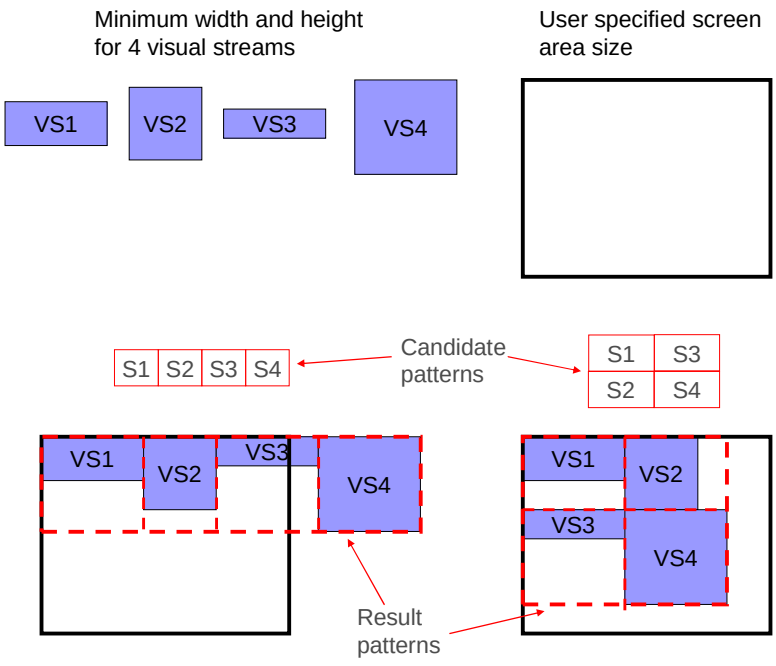

Figure 24. Illustration of testing resolution constraints on each candidate pattern

Note that the case for streams with highest priority (score is 3), we try to enlarge its share of the screen space first by enlarging its minimum width and height by a factor of 2. If the

algorithm manages to find a potential list for this, then in the final screen layout those high priority streams will have a larger screen space; if the algorithm fails, then the algorithm will restore the original minimum width and height of the high priority streams and try again ("go to start").

Step 3. Enlarge pattern to maximize screen area utilization

For each pattern P in potential_pattern_list {

Expand each subwindow in P uniformly so that P is of the same width and height as the given screen areas.

}

Explanation: in this step, we expand each subwindow proportionally so that we fully utilize the given screen area.

Example Walkthrough:

The goal of this step is to try to enlarge the patterns to fully occupy the screen area. The reason is that in step 2, we have been working with minimum width and height of each visual stream. Now that we are sure the remaining patterns will be able to host all the visual streams inside the screen areas, we want to expand these patterns to fully utilize the given screen space. For example, as shown in Figure 25, the 2x2 pattern is expanded to occupy the whole screen area, and each visual stream will occupy the entire subwindow correspondingly.

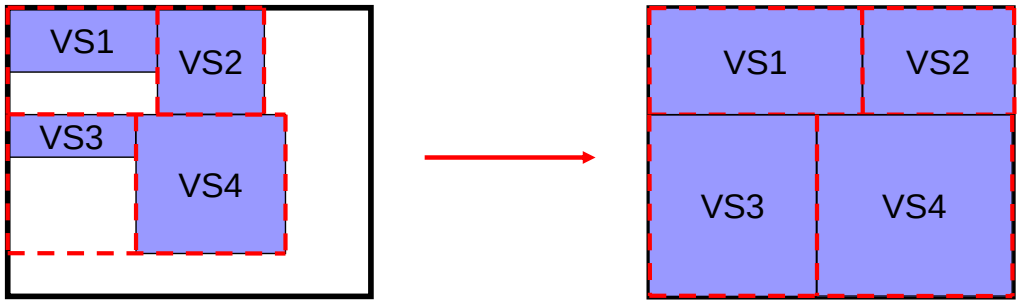

Figure 25. Illustration of How to Expand a potential pattern to occupy the whole screen area

Step 4. Compare potential patterns for maximum screen space utilization

For each pattern P in potential_pattern_list {

Total_utilization = 0;

For each subwindow SW in P {

Total_utilization += screen utilization of SW given the aspect ratio of the corresponding visual stream (note that for visual strams with no aspect ratio constraint, this utilization is its total allocated resolution).

}

Update maximum total_utilization

}

Output the pattern with maximum total_utilization.

Explanation: in this step we calculate the total space utilization for each pattern and select the pattern with the maximum utilization.

Example Walkthrough:

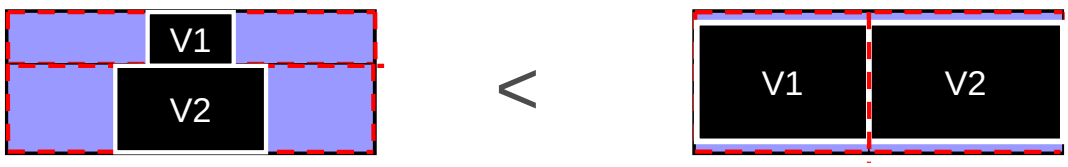

Figure 26. Illustration of effect of aspect ratio constraint on the screen space utilization

The goal of this step is to pick a single pattern out of all the potential patterns based on the total screen resolution utilization. In other words, this is to compare which patterns' subwindows fit best to the aspect ratio of the visual streams so that the least screen space is wasted. For example, for the two patterns shown in Figure 26 above, the black regions represent the videos played in the subwindow, and because of the aspect ratio constraint, the pattern on the right utilizes the screen space much better than the left one.

6.2 Summary

In this chapter, we have presented the screen layout manager that maps a set of *K* visual streams (selected by the visual director) onto the screen areas (specified by the user). There are many constraints on how this mapping should be done, such as the minimum width/height and the aspect ratio of each visual stream window, the number and size of the screen areas, and the output screen layout needs to be visually pleasing to the user (should not be fragmented window pieces on the screen). As a special case of the two dimensional bin packing problem, this problem is NP hard, and we have devised a heuristic algorithm that conducts an intelligent search on the constrained solution space for maximized screen space utilization.

Chapter 7. DISCUSSION

7.1 Other Potential Solution Frameworks

As discussed earlier, our approach heavily relies on the scores as a uniform measure that represents how much a viewer prefers to watch a visual stream. There are other solution frameworks that can be used to represent such a mapping from user preference to knowledge in computer. In this chapter, we will first discuss some of these alternatives and how we could direct visual stream switching based on them, and then compare between them and explain the tradeoffs we make in our choice in this project.

7.1.25 Bayesian Network

Bayesian network [3] has been widely used to describe probabilistic behaviors. Specifically, a Bayesian Network is a Directed Acyclic Graph, with the nodes representing random variables and the arcs between the nodes representing causal relations between the random variables. Nodes without any arc are assumed to be independent. In the Bayesian Network, the root nodes carry the absolute probability of each random variable, and the path from a parent node to a child node represents the conditional probability of the child node given the probability of the parent node. The probabilities of the nodes and arcs can be learned by observing the realized values of the random variables from a large set of sample data and also from prior knowledge. Once the probabilities are learned, the Bayesian Network can be used to infer the conditional probability distribution of the random variables given new observations.

To apply Bayesian Network approach to the problem of stream selection, we could treat the stream selection problem as calculating the probability P_{it} that, at time point t, the visual stream VS_i is interesting to the viewer. If the probability for a stream is higher than some preset threshold, i.e. $P_{it} \geqslant T$, we can then select that stream for screen display. As time goes by, the probability P_{it} for stream i ($1 \leq i \leq N$) will change according to the semantic events occurring in each stream, and so the stream selection may also be changed. This process is illustrated in Figure 27, where the probability of each stream *VSi* being interesting to the viewer P_{it} is calculated. Assuming a threshold of $T = 0.6$, the selection will change from {1} at time t to {2, 3} at time $t+1$.

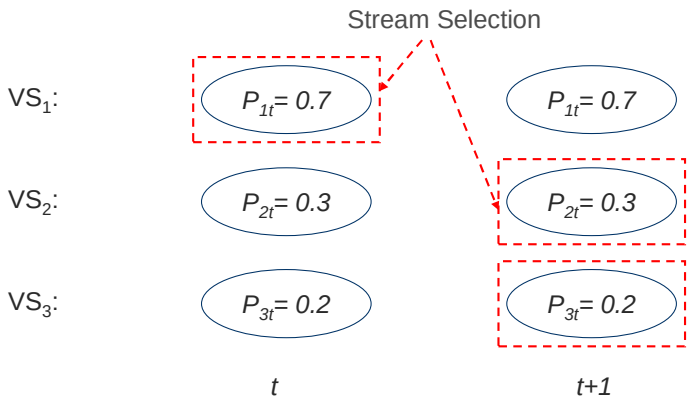

Figure 27. Example of Probability Based Stream Selection

The design of the Bayesian Network needs to reflect the causal relation between the observable facts (semantic visual/audio events) and the unknown inferences (i.e. viewer interests in each visual stream). A simple example Bayesian Network is shown in Figure 28 below.

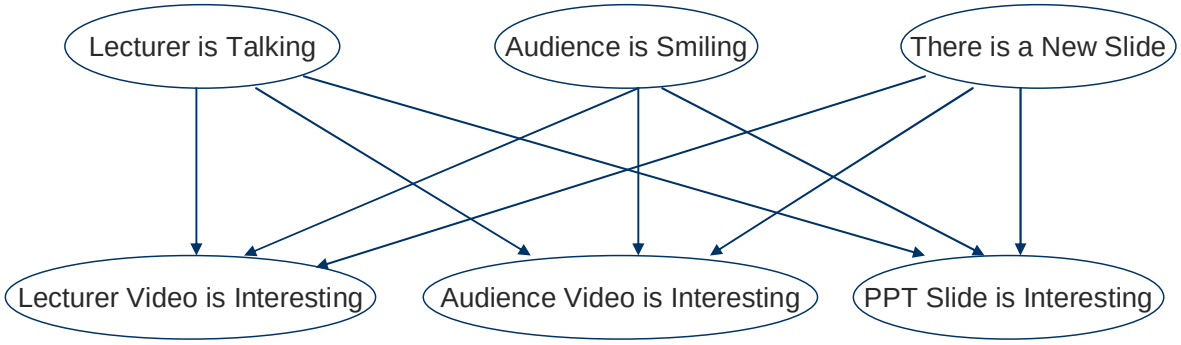

Figure 28. Example Bayesian Network to Calculate Stream Selection Probability

In this Bayesian Network, six Boolean random variables are represented by six nodes. The three root nodes represent three observable facts on three visual streams, which we assume can be automatically detected. The three non-root notes represent the three visual streams. Its value is *True* if the corresponding visual stream is interesting to the viewer and should be selected for screen display. Since the three non-root nodes represent relative preference of the viewer based on the current visual/audio cues of all visual streams, there is an arc between each root node and each non-root node, indicating their causal dependency. Note that the three root nodes are not connected since the corresponding semantic events are independent, and the three non-root nodes are also assumed to be independent. To utilize such a network to calculate the probability of each stream being interesting to the viewer, we can apply the following formula:

$P(VS_1 \text{ is interesting}) = P(VS_1 \text{ is interesting} \mid \text{audience is smiling}) * P(\text{audience is smiling}) + P(VS_1 \text{ is interesting} \mid \text{lecturer is talking}) * P(\text{lecturer is talking}) + P(VS_1 \text{ is interesting} \mid \text{new slide}) * P(\text{new slide})$

In practice, to build such a Bayesian Network for live visual streams switching, we need to first learn the probability parameters in the network in the following steps:

1. Collect sample video data with viewer's judgment. We will need to invite a large number of viewers to watch the same visual stream and specify which visual streams are interesting and should be selected for each time period. We also need to record all the visual/audio events observed on all the visual streams.
2. Learn the root node probabilities. This can be computed by the ratio between the amount of time each root node is *True* and the total length of the video. For example, if the playback is 100 seconds in total, and the event "lecturer is talking" has been active for a total of 70 seconds, then the probability of the root node "Lecture is Talking" is 70/100 = 0.7.
3. Learn the conditional probabilities from the root nodes to the non-root nodes. This can also be easily calculated from the observed sample data. For example, if over the 70 seconds that lecturer is talking, on average for 50 seconds the lecturer video is marked as interesting by the viewers, then the conditional probability *P(lecturer video is interesting | lecturer is talking) = 50/70 = 0.71*.

Once we have learned the Bayesian Network from the observation on viewer subjects, we can assume the same probability distribution will also apply to new lecturing sessions and use the probability to make stream selection decisions. For example, any time during a lecture, we can deduce the probability that each of the visual streams is interesting to the viewer, and select the streams with a probability higher than the threshold. Whenever a new visual/audio cue is observed, such as the lecturer starts to talk or stops talking, the probability will be recalculated, which might result in a new stream selection (and so a stream switching).

7.1.26 SEEV Model

Since we have already covered the SEEV model in Section 4.2, we will mainly discuss how we can apply the SEEV model to approach the stream switching problem in this Section.

The SEEV model specifies the probability of visual attention being allocated to a certain *area of interest* **A**:

$$P(\boldsymbol{A}) = s\boldsymbol{S} - ef\boldsymbol{EF} + (ex\boldsymbol{EX})(v\boldsymbol{V}).$$

Therefore, to use the SEEV model to determine stream selection and screen layout, we need to learn the property vector (**S**, **EF**, **EX**, **V**) for each visual stream and also the weight parameters (*s, ef, ex* and *v*), and then use the given formula to calculate the probability for whether each visual stream should be selected on the fly as the lecture session evolves.

To learn the property vector of each stream, we can define how each property is computed from the characteristics of the visual stream. For the salience property **S**, we can define it to be proportional to the color contrast of the visual stream image to the other visual streams on display. We could compute the average brightness of the two and take the difference **D**, and then set **S** = *d* * **D**, where *d* is a tuning parameters set by experiment. For the effort property **EF**, we can assume that the effort a viewer makes to view the visual stream is proportional to the minimum resolution of the visual stream, and the larger the minimum width and height of a stream is, the more efforts the viewer's eyes have to make to capture the visual information in that stream. Formally, we can set ***EF*** = *w * minimum_width * minimum_height*, where *w* is a tuning parameter. For the event expectancy property **EX**, it can be an average rate of events that is occurring to a stream. That is, ***EX*** = *e * average_number_of_event_per_second*, where

e is a tuning parameter. Finally, for the value property **V**, it can be assigned by the viewer as a customization tool, similar to the score in our approach.

Once we know how to compute the property vector for each stream, the weight parameters in the SEEV model formula can be trained in the following way. We could invite a large number of viewers to watch the playback of a lecture recording. All the visual streams will be presented in a tiling screen layout, and we can use an eye-tracker to track where each viewer looks while he watches the playback. With the eye-tracking log, we can derive the probability that each visual stream is attended by the eye, and then compute the weight parameters accordingly.

After the model parameters are acquired, we could make on-line stream selection decisions using the output of the model formula. Specifically, the probability is updated periodically, such as once per second, and the streams with high probabilities are selected for on screen display. Of course, some timing mechanisms (minimum/maximum on screen time) may be applied similar to our approach so that the visual quality is acceptable.

7.1.27 Comparison between Different Solution Frameworks

Now we have three solution frameworks available to solve the stream selection problem: our score-based solution, the Bayesian Network based solution and the SEEV model based solution. We will discuss the main differences between them in the following aspects.

7.1.27.1 Key Assumptions

Since viewer preference among all the visual streams is a very subjective matter, all the three models make some strong assumptions. In the Bayesian Network model, the key assumption is that there is a causal relation between the visual cues observed (i.e. lecturer is

talking, audience is smiling, etc) and the underlying viewer interest. This assumption may not hold if there are semantic cues beyond what the computer can recognize that are triggering changes in viewer's interest. For example, maybe a human video switcher will switch out of the lecturer's video because his facial expression is boring to watch, this can hardly be learned by computers. If that is the case, the parameters of the Bayesian Network will not reflect the true probability distribution. In addition, Bayesian Network also assumes that nodes without arcs are probabilistically independent. However, this may not be true since the semantic cues may be dependent on each other. For example, a "questioning" event observed in the lecturer video may trigger a "laughing" event in the audience video.

For the SEEV model, a key assumption is that the four properties will completely cover all possible reasons why a viewer wants to switch to a visual stream. However, this may not be true. For example, if a viewer has been watching the lecturer's video for too long, he might want to switch to another video that does not have a high probability if computed using the SEEV formula. This actually represents the fact that the value property $\boldsymbol{V}$ of a visual stream may change over time, and such changes may be triggered not by external stimulus to the viewer's eyes, but changes in the mind of the viewer.

For our score-based framework, the key assumption is that the viewer is consistent in their stream preference under the same situation. For example, if the user sets the score of the lecturer video's "talking" state to be of the highest score 3, then we assume that, before the user changes the score setting, that "talking lecturer" is always taken as high priority no matter what happens in other visual streams. In reality, this applies well to the simple scenarios such as lecture presentations, but for more complicated scenarios, such as soccer game broadcast, where the viewer's preference may change very frequently, the viewer may not be able to

change the score setting in time or on every preference change, and the assumption will be violated. In such cases, automatic learning of user preference changes may be required, as discussed later in this Chapter.

7.1.27.2 Scalability

Essentially all three solutions rely on the state of all the visual streams and the visual/audio cues detected, so at any time t, the stream selection is dependent upon the state combination of all visual streams. Such state explosion may cause scalability problems. Here, we calculate the time and space complexity for each method.

For the SEEV model, assuming we work with N visual streams, then $4N$ property parameters need to be calculated every period (e.g. every second). So the computation load over a duration of T seconds will be $O(N*T/t)$. In terms of space complexity, only the property parameters need to be stored, which is $O(N)$.

For the Bayesian Network model, assuming there are N streams, each is observed to have an average of K visual/audio events, then there will be $N*K$ number of root nodes, N non-root nodes, and $O(N^2K)$ number of arcs connecting the root nodes and non-root nodes in the network. Therefore, the space complexity is $O(N^2K)$. During a live session, each time a new visual/audio event occurs, the probability of viewer preference for all the N root nodes in the Network will need to be recalculated. Assuming during a T second duration, there are E such events, then the computation will be $O(NE)$.

For our score-based approach, one of its strength is that it avoids state explosion by simplifying the state information to an integer score. For example, if two streams are currently having scores of *3* and *2*, it may actually represent many possible combinations of states of the

visual streams. Such information is hidden from the stream selector, who only works with a uniform array of scores. More concretely, assuming there are *N* streams, each is observed to have an average of *K* visual/audio events, then we only need to store the score for each stream's each state, which is *O(NK)*. And the computation is done only when a new event is detected, so assuming during a *T* second duration, there are *E* such events, then the computation will be *O(NE)*.

7.1.27.3 Level of Viewer Participation

Since viewer's subjective preference and semantic knowledge about the visual streams are key determinant to the stream selection, all the three models include the viewers in the decision loop to learn the knowledge from them. However, they are different in the level of viewer participation as follows.

For the Bayesian Network model and the SEEV model, the user does not actively participate in the knowledge gathering process. He or she only needs to watch a few sample sessions, and the system will extract knowledge from his/her eye movement and the accompanying visual/audio cues. On the other hand, our solution requires a proactive reviewer (the system administrator or video supporting staff) to participate in the configuration of the system before or during a live presentation session.

The tradeoff between these two approaches can be discussed in two aspects. First, since the user is actively involved in the new session, he/she can adjust the knowledge mapping just-in-time to reflect his/her most current preferences. On the other hand, for the passive training approach, the knowledge learned from the viewer's eye movement traces only represent past viewer preferences on past presentation sessions, which may be less suitable to direct the

stream selection for new sessions. Second, the active user participation does entail a training process before the user can accurately setup the system parameters to freely express his/her preferences.

7.1.27.4 Incorporation of Cinematography Rules

As we introduced earlier, there has been a large body of existing work on cinematography rules, which specify how to capture a scene with multiple cameras so that the visual messages is most efficiently delivered to the audience. Needless to say, it would be desirable to incorporate such professional knowledge into the visual stream management framework.

In this aspect, our approach uniquely wins because it allows the user to setup transition idioms. Such idioms allow the user to apply cinematography rules in combination with the score-based solution. Such rules are converted to deterministic transitions in the timed automaton, and they have higher priority than the automatic stream selection based on pure score comparison. For the other two solution frameworks, it would be much harder to incorporate such professional rules into the model because they are not based on a state machine type of knowledge representation.

Overall, all solution frameworks have their strength and drawback. We have chosen to adopt the score-based framework since it makes a reasonable assumption about viewer's preference changes, it is scalable in computation and storage complexity for large number of visual streams and large number of visual/audio cues, it includes active human participation for customization of viewer preference in a timely manner, and it seamlessly incorporates existing work of cinematography rules.

7.2 Learning User Preference for Automatic Configuration

The current system requires users' active participating in the stream selection decision process. Specifically, the score configuration, the timing constraints and the transition idioms are all assumed to be input by the user based on their preference. Although a set of default parameters are provided that will ensure a default screen layout management scheme without any user input, the user has to change these parameters to customize the screen output based on his own preference. Although such a design may suffice simple usage scenarios like distributed classroom, it may lead to a key limitation of the system when the usage scenario becomes complex (and so the parameter setting). For example, for live broadcast of a soccer game, there could be potentially 30 video camera streams covering different view aspects of the soccer field and the players, and each stream may have multiple semantic states depending on different types of visual/audio events happening to it. In such situations, the system configuration (score, timing constraints and transition rules) will become more complicated, and it would become a challenging task even for professional video supporting staff to configure the system parameters correctly to achieve the desired screen output.

In this Section, we discuss a possible solution for automating the configuration parameters by learning from user input. As shown in Figure 29, currently the user is a key component in the decision loop. Initially the user setup his preference at the user layer, including the interest score, timing constraints and the transition idioms. Then these parameters are translated into a timed automaton, which is updated whenever the user makes any changes to the parameters. During a presentation session, the timed automaton is executed

to make on line stream selection decision based on the state of all visual streams. The list of selected streams is passed to the screen manager, which computes the screen layout and position the visual stream windows properly. As the user watches the screen output, he can make changes to the configuration parameters to indirectly affect the screen output.

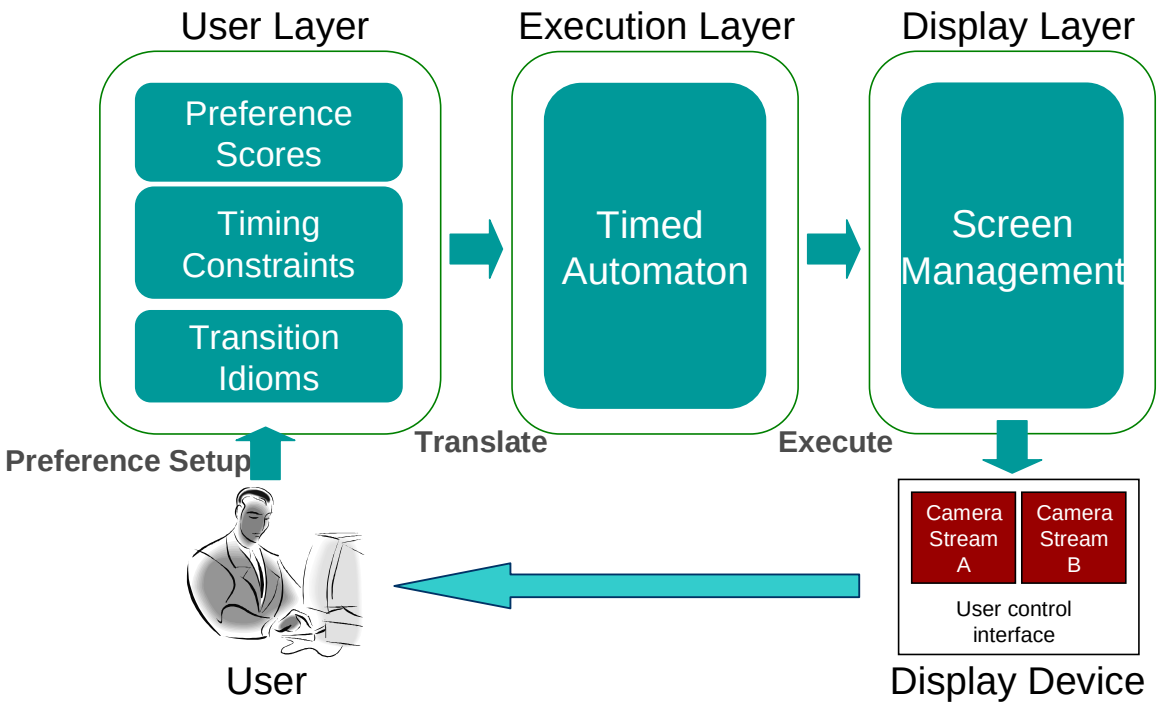

Figure 29. Overview of Our Stream Selection Decision Loop

What we would like to do is to let the computer system observe the user's input and the corresponding screen output, and learn the parameter configuration automatically. To achieve such a goal, we need a systematic method to train the computer for all the configuration parameters. One possible direction to explore is described below.

7.2.28 Learning Score Setup

Score setup refers to the assignment of score for each visual stream's each state. For a distributed lecture with *N* streams and *M* states per stream, the total number of scores to setup is *N*M*. For example, for a four way class, if there are five video streams, each with a "talking" state and a "silent" state, then *5 * 2 = 10* scores need to be setup.

Since the score is used by the virtual video switcher to compare the interestingness or importance between visual streams, only the relative order of the scores is important. Suppose we have collected a large set of video switching samples done by video switching professionals, we can learn the relative order of all pairs of stream state combinations in the following way. For visual stream *v1* and *v2*, suppose *v1* has states *s11* and *s12*, and *v2* has state *s21* and *s22*, then for the state combination (*s11*, *s21*), we can collect the statistics that out of the total number of time points when *v1* is in state *s11* and *v2* is in state *s21*, what is the percentage of time *P* that *v1* is selected over *v2*. If *P* is greater than a threshold T_{high} (such as 0.7), then we can conclude that *s11* should have a higher score than *s21*. Similarly, if *P* is less than a threshold T_{low} (such as 0.3), we can conclude that *s11* should have a lower score than *s21*. If *P* falls between (T_{low}, T_{high}), then they should be assigned the same score level. Similarly the relative score level can be learned for all the other state combinations (*s11*, *s22*), (*s12*, *s21*) and (*s12*, *s22*).

Once we have learned the relative ordering of all possible state combinations, we can start trying to assign score integers to each stream's each state that satisfy the ordering. For example, If we have learned that *S11 < S21 < S12 = S22*, then we can assign scores in this way: *S11 = 1, S21 = 2, S12 = S22 = 3*. However, when the number of streams and semantic states of each

stream grows larger, there may not be a score assignment that satisfies all the relative ordering. This is due to the inherent limitation that a single score is used to represent the viewer's evaluation of a stream state's importance, while the sample switching from the video switching professionals may exhibit much more information and changes in preference. Three directions can be explored to solve this problem. One is to increase the scale of the scores, such as from 4 levels to 100 levels, which will increase the chance of being able to find a satisfying score assignment plan. Another potential solution is to relax the requirement on that score assignment and adopt a score assignment that satisfies the most number of ordering relations. Finally, score assignment can be combined with transition idioms to better match the switching result from video switching professionals.

7.2.29 Learning Timing Constraints

As described earlier, the timing constraints refer to the minimum and maximum on screen time for each visual stream's each state. This can be very easy to learn from samples video switching. Given a relaxation threshold T_{relax}, such as *0.9*, we can find out the minimum and maximum number of seconds of on screen time for T_{relax} percentage of all the time periods between all video switching.

7.2.30 Learning Transition Idioms

In our framework, transition idioms are used by the user to override the stream selection resulting from pure score comparison. In fact, if the score assignment is perfectly consistent with user preference, there will not be any need to configure any transition idioms. However, in reality, score comparison alone may not be enough, especially for complicated scenarios with a large number of streams and multiple states. In such cases, the transition idioms

learning can be combined with the score learning process. Specifically, we can find out the score assignment plan that satisfies the most number of relative ordering of stream state combinations, and then for the remaining orderings learned from video switching professional's samples, we can add transition idioms one by one to match those special cases. For example, suppose we have learned the ordering that *s12 < s11, s11 < s21, s21 < s22, s22 < s11*, which can never be satisfied, we can assign the scores like this *s12 = 1, s11 = 2, s21 = 3, s22 = 4*. Then for all the cases where *s11* is selected over *s22*, we add a transition idioms saying that when such state combination occurs for *X* seconds, we will select *s11* over *s22* despite the score comparison result.

7.3 Virtual Memory Management and Visual Space Management

Virtual memory management has been a classic topic in computer science. The problem arises when the amount of memory that can be accessed very fast is less than the total amount of data and code on the disk that are to be accessed. Basically a virtual memory hierarchy is established that assigns a virtual memory address to all data while only a portion of the data is actually loaded into the physical memory. The key of virtual memory management is when and how to replace existing data blocks in the physical memory with new data blocks on the disk so that overall the most important data blocks are kept inside the physical memory when they are needed. From this perspective, there are some very interesting similarities between virtual memory management and visual space management, since visual space management aims at assigning the physical display space to a small subset of all visual streams. In this section, we compare between them from the following four perspectives:

- When is data replaced

  Both virtual memory management and visual space management exhibit the “on-demand” characteristic in terms of data replacement. With virtual memory, the data replacement is done when there is a memory access miss, i.e. the data needed is not found in the physical memory. With visual space management, the new streams are selected for display when some interesting semantic events have occurred, making some visual streams more interesting to watch than the currently presented streams. In this sense, the assignment of the valuable yet limited resource (physical memory or screen space) is changed when the priority among all the available content (data on the disk or all visual streams) is changed due to some critical events (new data needs to be access or some semantic event occurs)

- What data is being replaced

  Due to the limited availability of space (physical memory or visual space), when new data is being imported, some existing data may need to be replaced. In both cases, the replacement is done by comparing the “value” of all data currently occupying the limited space. Specifically, for the case of memory management, a most popular replacement policy is to replace the least frequently used data blocks with the rationale that such blocks are least likely to be valuable again to be brought back the memory. For the case of visual space management, the stream selection process will compare the scores of all the visual streams, and the streams with lower scores will be replaced by the new (higher scored) streams.

- Working Set

  An important concept with virtual memory management is working set, which refers to a minimum set of data blocks that are very frequently accessed. If the total amount of physical memory is less than the size of the working set, then threshing (very frequent and repeated data replacement) will occur. Similarly, in visual space management, there may also be a working set of visual streams that are frequently triggering viewer interest and should be displayed. If the available visual space is less than the total screen space needed to present the working set of visual streams, then undesirable threshing will occur. For example, for a distributed lecture, the video of the lecturer and the PowerPoint slideshow may always be of high importance. If the available screen space is not enough to present these two visual streams at the same time event at their minimum width and height, then they will be shown on the screen alternatively, replacing each other repeatedly. This should be avoided when setting up the screen area allocation.

7.4 Summary

In this chapter, we have discussed several important issues regarding the design decisions for our visual stream management framework. We compared our score-based framework with other solutions, automating user preference parameter configuration, and drew similarities between viual space management and virtual memory management. These complete the discussion on the MyView system design, and we will move on to the evaluation part of MyView in the next chapter.

Chapter 8. EVALUATION

8.1 Overview

We have divided the evaluation of our solution into three main tasks:

Task 1 is to verify the functional correctness of the timed automaton based stream selection algorithm. As introduced earlier in Chapter 5, we have developed the algorithm to generate the full timed automaton and verify the desired properties using the UPPAAL toolbox.

Task 2 is to evaluate the screen layout calculation algorithm. We simulate the scenario of a 7-stream distributed presentation session, and applied the screen layout calculation algorithm for all possible selection combination of these streams.

Task 3 is to study the usability of our solution. We have built a distributed presentation recording/broadcast system that utilizes our solution for visual stream management. We will discuss how we have evaluated our solution through a mid-scale user study.

8.2 Task 1 – Stream Selection Algorithm Evaluation

In this experiment, we want to evaluate the timed automaton used for stream selection process for its **functional correctness** in terms of liveness and state reachability.

We used the algorithm described in Chapter 5.3 to automatically generate the full timed automaton in the syntax of the UPPAAL verification toolbox, then we used both the *Simulator* and *Verifier* provided in the UPPAAL toolbox to evaluate the correctness of the automatically generated timed automaton.

8.2.31 Complexity of Timed Automaton

As already analyzed in Section 5.3.3, the complexity of the full timed automaton is $O(M^N)$, growing exponentially with the number of visual streams. Using the automatic timed automaton generation algorithm given in Section 5.3, we generated a set of timed automata with the parameters shown in Table 8-1.

Parameter	Value	Comments
Number of streams K	$2 \leq K \leq 6$	
Number of shot types	$2 \leq M \leq 4$	
Idiom matching rate	2%	2% of the states have a corresponding matching idiom
Minimum on screen time	Random 10 and 30	
Maximum on screen time	Random between 30 and 100	

Table 8-1. Parameter Configuration For Automaton Generation

We generated *3* timed automata or each *K* - *M* combination to get an average value of three measurements: *time to generate the automaton*, *number of states* in each timed automaton, and *number of transitions* in each timed automaton. The results are shown in Figure 30 below. Specifically, the top figure shows the exponential growing time it takes to generate a full automaton with increasing number of *M* and *K*; the middle figure shows the exponentially growing number of states in the resulting automaton; and the bottom figure shows the exponentially growing number of transitions. Note that all the three graphs' Y axis has been preprocessed for its Log_{10} value to make the graph easier to appreciate.

From the graphs we can conclude two issues. First, the complexity of the timed automaton is forbiddingly high even for automatic processing. Therefore, it is too complex, if not impossible, for human administrators to manually authorize such automata when the number of visual streams grows larger. Second, we cannot generate the full timed automaton at runtime whenever there is a change in user interest parameters. Therefore, we have to resort to the on

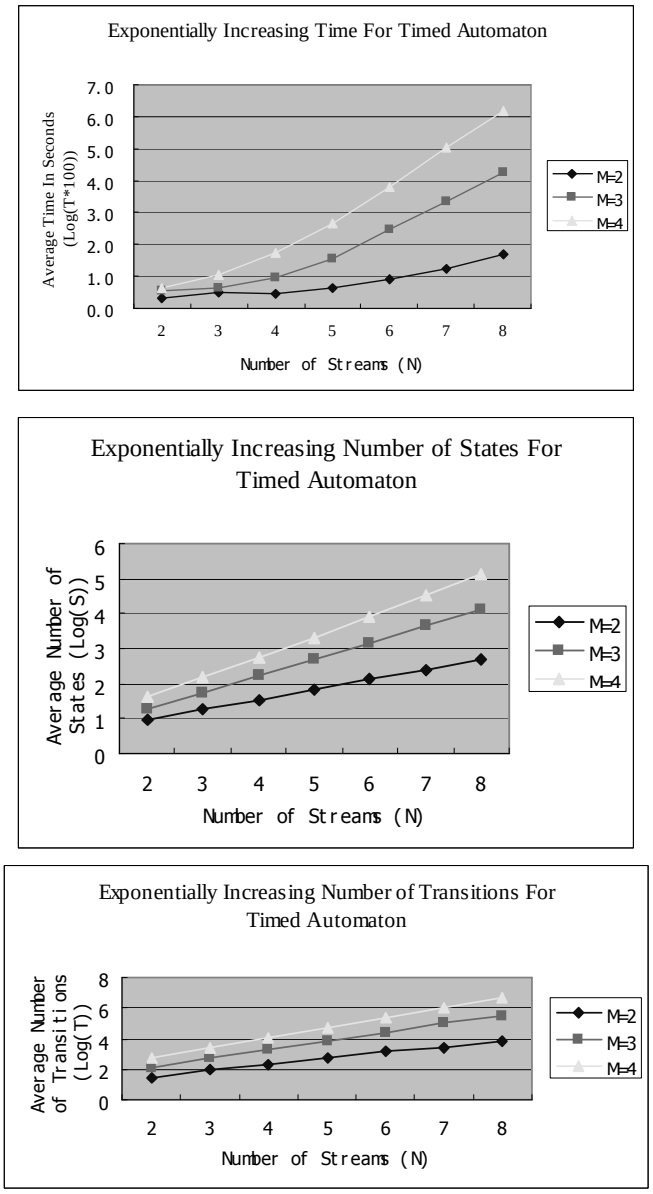

Figure 30. Complexity of Automatically Generated Full Timed Automaton

line algorithm described in Section 5.2 that only maintains the current state of the timed automaton and the potential transitions.

8.2.32 Results from UPPAAL

The UPPAAL simulator is a tool for visually verify the functional correctness of the automaton, and it works in the following way. Given a timed automaton system file in UPPAAL specified format, the simulator will load the system into a graphical user interface, and the format of the automaton is examined against the UPPAAL grammar. Any undeclared variables, such as states, transitions or channels, will be identified. If the loading is successful, the user can start the simulation of the run time execution of the system, including the state transition of all the automata in the system and the current output of the automata. The clocks are simulated in the sense that the time between one transition and the next is skipped, so the simulator continuously moves through the state transitions at a constant speed. Note that for the simulation process, the change of shot type for each visual stream is simulated by randomly determined change of state of the visual stream automaton. During the simulation, the current state of each automaton is marked with red color, and the enabled transitions and the current output are also listed on the user interface. Because of the non-negative clocks used in the timed automaton, the system can evolve infinitely. When the user terminates the simulation, the trace can be saved.

Since the simulator has to be manually operated, we cannot simulate a large number of timed automata. Therefore, we have randomly chosen 5 of the timed automata that are generated during the run time of MyView to test with the simulator. The result is that all the selected timed automata can be successfully loaded and executed in the simulator for over 2 hours without deadlock or system failure. Figure 31 is a screen shot of the simulator interface during a simulation.

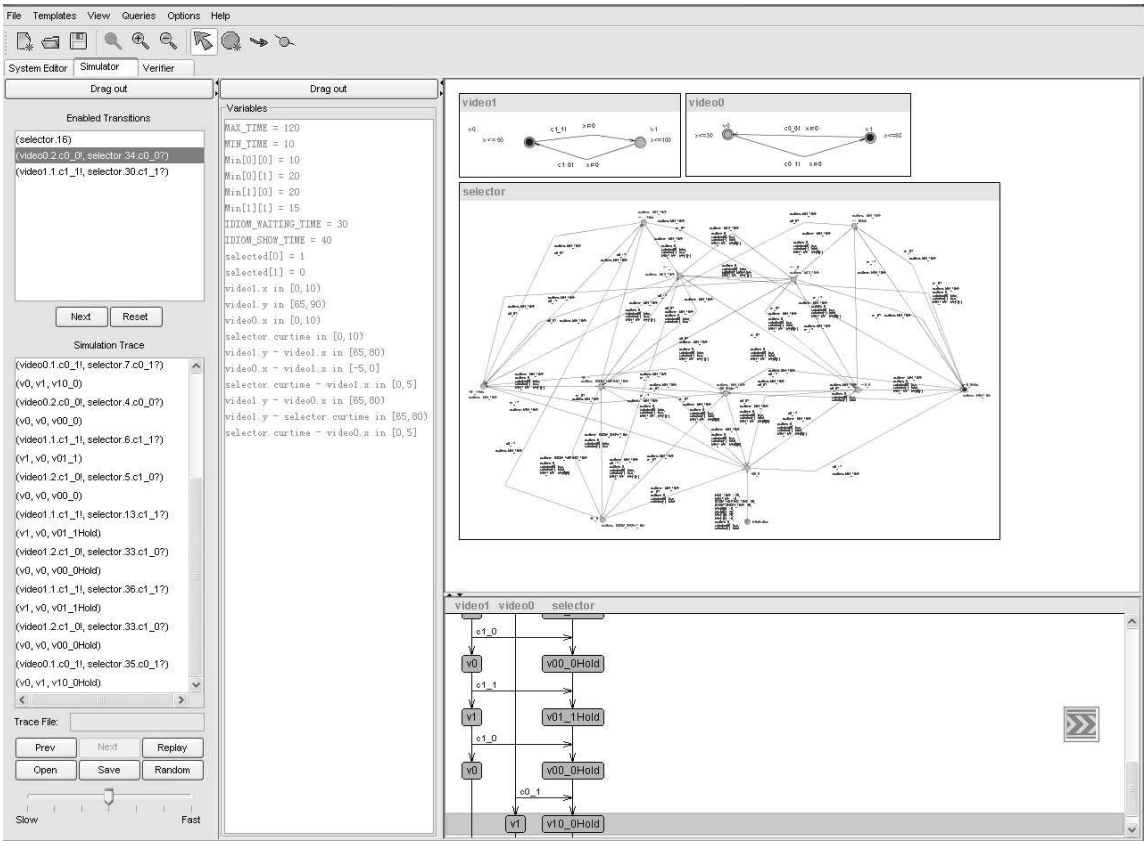

Figure 31. Screen Shot of Simulator Interface

We have also used the UPPAAL verifier tool to analyze the timed automaton in terms of *functional correctness*. The verifier tool checks the *invariance* and *liveness* properties of the timed automaton against a list of *queries* by "on-the-fly exploration of the state space of a

system in terms of symbolic states represented by constraints" (quote from UPPAAL Help). Specifically, we used the following two types of queries:

- Liveness query in the format of "*A [] not deadlock*", which means it is an invariant claim that the system is deadlock free.
- Reachability query in the format of "*E <> target_state*", which means there exists a path from the starting state to the *target_state*. The *target_state* may be "*selector.v01_1*" to represent the *v01_1* state of the stream selection automaton named "*selector*".

Again, because the verifier tool can not be automated for model checking on a large number of automata, we chose to test the 5 selected timed automata generated in run time. Each of these automata is put through the two types of queries. For the reachability test, we only chose 5 random states from each of the 5 selected automata to verify whether those states are reachable from the initial state.

The result is that all the automata are deadlock free, and each state of each automaton is reachable. Figure 32 shows a screen shot of the user interface of the verifier.

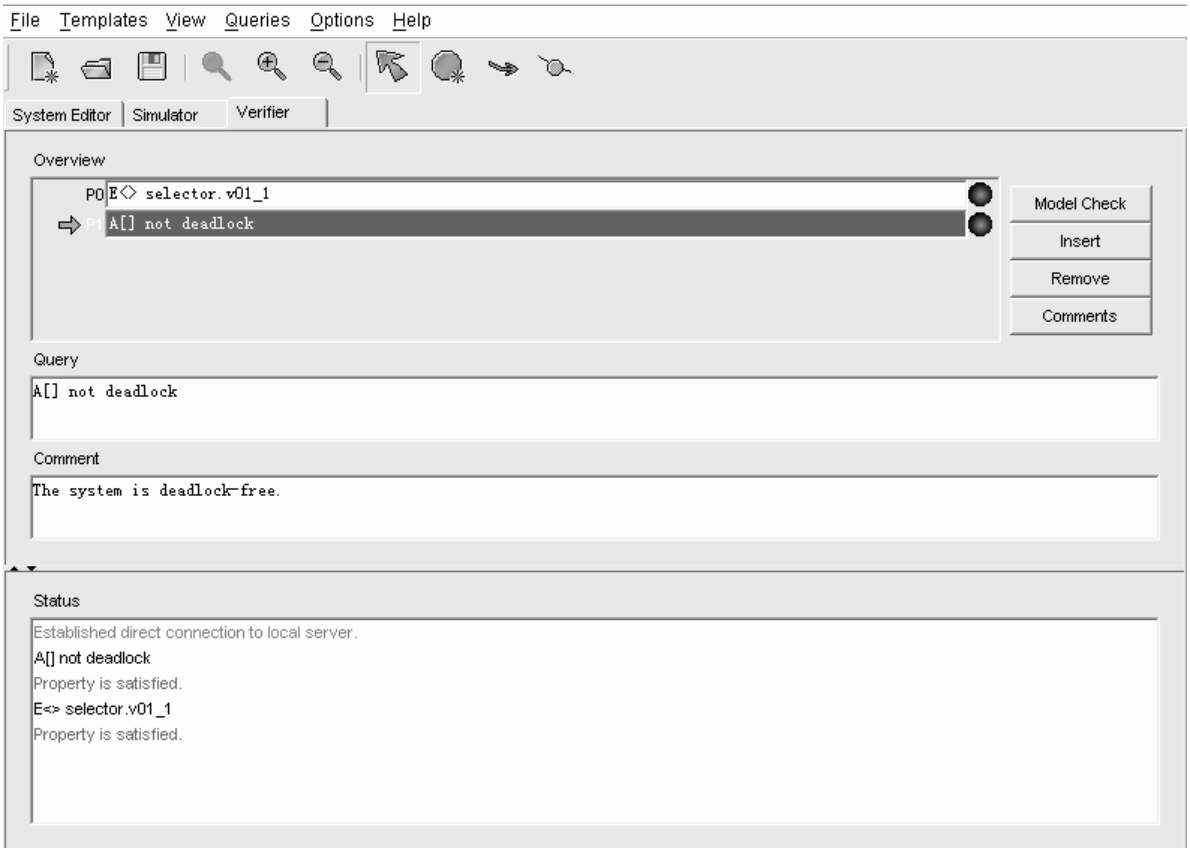

Figure 32. Screen Shot of UPPAAL Verifier for Model Checking

8.3 Task 2 – Screen Layout Calculation Algorithm Evaluation

To test the screen layout calculation algorithm, we simulated the layout calculation algorithm on 7 input visual streams and 3 types of screen area scenarios. Specifically, we assume there are 7 visual streams in the application session with the resolution constraints given in Table 8-2 and three screen area size assumptions given in Table 8-3:

Visual Stream Name	Resolution Constraint	Visual Stream Name	Resolution Constraint
Video camera stream 1	320x240	Web browser	800x600
Video camera stream 2	160x120	Whiteboard	400x300
Video camera stream 3	160x120	Text Chat	200x400
PowerPoint slideshow	600x400		

Table 8-2. Resolution Constraint Assumptions for Automaton Generation

Scenario Name	Single Full Projector Screen	Single Full Handheld Screen	Double Screens
Screen Area Size	1024x768	800x600	1024x768 + 800x600

Table 8-3. Screen Area Size Assumptions for Automaton Generation

We generated all the possible selection combinations out of the 7 visual streams, and for each screen area scenario, we removed those "oversized" combinations where the sum of the pixel area of the selected visual streams is greater than that of the screen areas. Then we tested how many of the remaining "non-oversized" selection combinations can be successfully arranged by the screen layout calculation algorithm onto the screen areas.

On average, the screen layout calculation takes $2.8 * 10^{-5}$ seconds, so it is very lightweight and can be used for real time screen layout decisions. Figure 33 shows the results. We vary the number of selected stream from 1 to 7, and see for each screen size scenario the maximum number of supportable stream combinations. For example, for the single screen area 1024x768 scenario, if only 1 stream out of the 7 is selected for display, then all the 7 combinations can be accommodated by the screen area. However, if 2 out of the 7 streams can selected, then out of the total 7x6/2 = 21 combinations, only 13 of them can be accommodated.

Several observations can be made. First, the larger the available screen areas are, the more visual stream combinations can be supported, which is intuitively correct. Second, the more selected visual streams, the less likely that a screen layout plan can be found, given the stringent requirements on the screen layout. Therefore, when the user does not allocate screen areas that are large enough, then when multiple streams are selected, those with lower scores will be dropped by the screen layout calculation process.

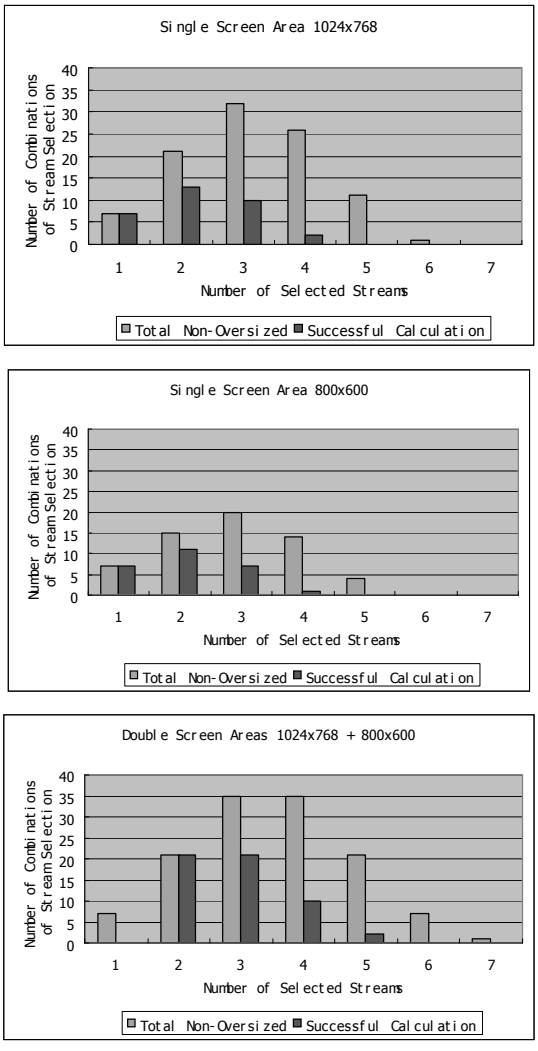

Figure 33. Success Rate of Screen Layout Calculation Algorithm

8.4 Task 3 – User Study Using MyView System

8.4.33 Evaluation Goal and Metrics

Since the visual space management problem focuses on user interest and customization, the user study is a very important tool for evaluating the success of our work. Specifically, we hope to evaluate our solution with the following goals:

- **Ease of Use**: we want to study whether the MyView system is easy to use, especially considering that users need to setup and change their preferences during a presentation session.
- **Value of Customization to Users**: we want to know whether users would prefer the function of customizing the presentation based on their interests.
- **Quality of Stream Selection**: we want to evaluate the quality of visual stream selection for whether the automatic selection will match the user interest.
- **Quality of Screen Layout Calculation**: we also want to know whether the users will like the visual stream arrangement on the screen that is automatically computed.

Overall, the user study has been divided into five phases as shown in Figure 34. In the *Preparation* phase, we recorded a presentation with audiences participating both locally and remotely, and this recording was used throughout the experiment for visual stream management. In the *Interview with Video Switching Professional* phase, we invited video switching professionals to manually conduct switching during a real-time playback of the recorded presentation, and the switching log was taken as a benchmark for comparison with

automatic switching in the later phases. In the *Comparison Study* phase, a focus group of non-professional user subjects were invited to watch and compare the screen output that is switched by (a) the automatic visual stream manager and (b) the video switching professionals. Then in the *Manual Switching* phase, the focus group was asked to manually conduct visual stream switching by picking the streams that they think should be switched onto the screen. The switching log acquired in this phase was combined with the log from the professional switcher to serve as a prior knowledge on which visual stream segments were important, which was used to evaluate the automatic visual stream manager's switching capability. And finally, in the *General Comments* phase, further user feedback was acquired from a questionnaire.

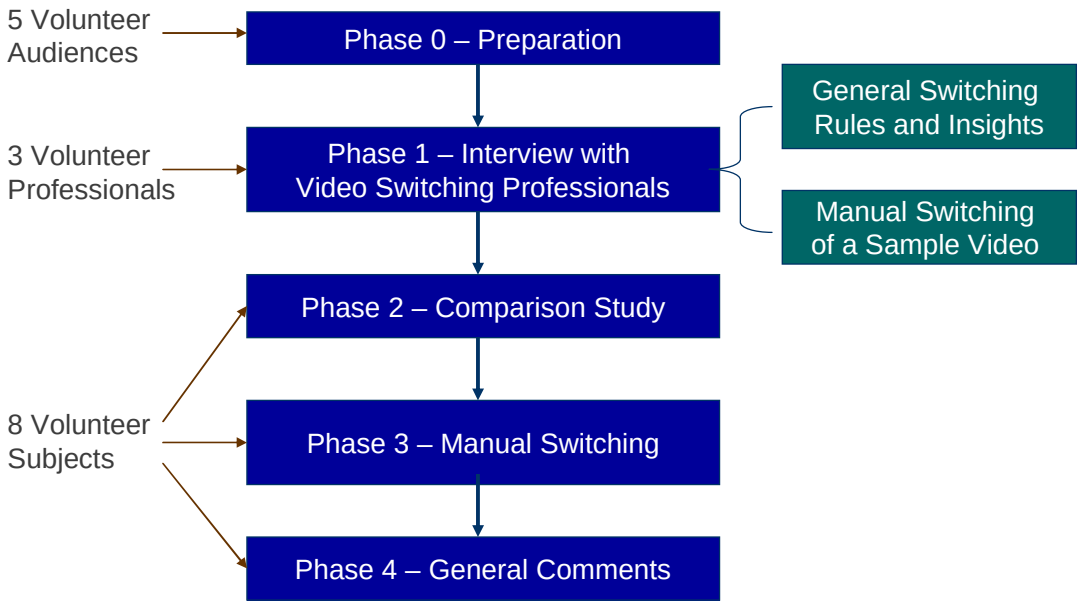

Figure 34. Five Phases of the User Study

In terms of evaluation result measurement, we adopted the following metrics.

- **User Satisfaction Factor**. We hand out a questionnaire at the end of the user study asking the user whether they were satisfied by the automatically switched video output (that is, they would love to use the system when watching a remote presentation). The User Satisfaction Factor is calculated as ***number of satisfied users / total number of users***.
- **Stream Selection Matching Ratio**. This metric measures how much of the stream segments selected by human (video switching professionals and user study subjects) is also selected by the computer. For example, for a 100-second long presentation session that is captured by 4 cameras, there are totally 100*4 = 400 *time points* (per second per stream). The matching ratio is calculated as the percentage of time points that are selected by the computer out of the number of time points selected by the human users. If the human user totally selected 150 time points for display while 120 of them are also selected by the computer for display, then the matching ratio is 120/150 = 80%. One difficulty in the measurement is to determine a ground truth of the exact time points that are selected by the human users since different users have different selections. Therefore, we define a parameter called *Overlapping Ratio* that measures, for each time point, the percentage of users who agree that this time point is important and should be included as the ground truth. By increasing the Overlapping Ratio, we can get a consensus time point selection that is agreed upon by more users. More details of the metric calculation will be given below in the result discussion.

8.4.34 Phase 0 – Preparation

To prepare for the user study, we recorded a short 15-minute presentation. The room setup is shown in Figure 35 below.

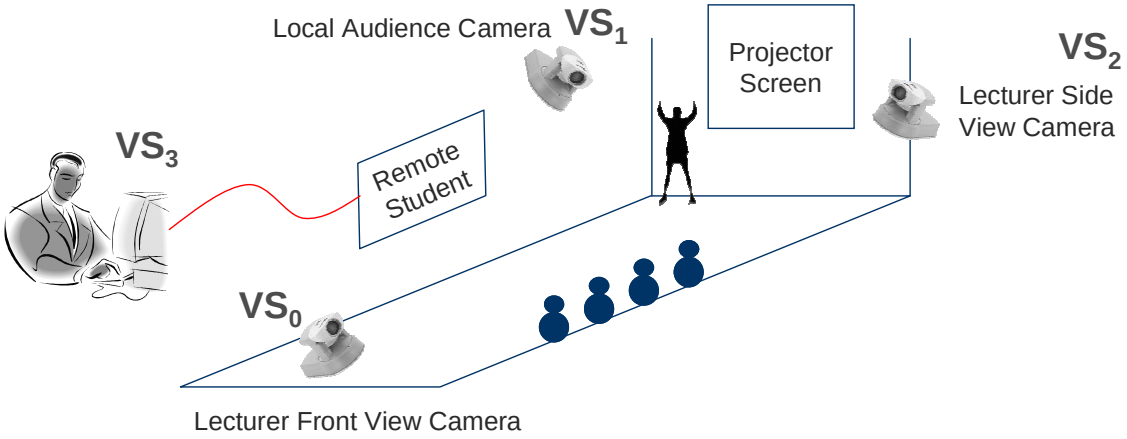

Figure 35. Visual Streams for Presentation Recording

One lecturer presented against a PowerPoint slide shown on the projector screen, and four audiences sat facing the lecturer. Three IP cameras (we adopted Axis 2130PTZ network cameras) were used: one to capture the front view of the audiences (VS_1), one to capture the front view of the lecturer (VS_0), and one to capture the side view of the lecturer when he turns back to face the projector screen (VS_2). One remote audience participated from her desktop, and her face was shown on a plasma display (VS_3). The voice of the lecturer was captured using a headset microphone, and the voice of the audiences in the room was captured using a ClearOne room microphone. The presentation contained both monologue of the lecturer and conversations between the lecturer with one or multiple audiences. All five audiences spoke during the presentation for various lengths of time.

In terms of experiment setup, we have built the MyView system on top of the ConferenceXP Platform. The user interface code is written in C#, and the camera control and

voice detection parts are implemented in Visual C++. The system runs on Windows Platform, and the PCs are configured with 3.2GHz Processor, 1.5G Memory and 80GB hard drives.

8.4.35 Phase 1 – Interview with Video Switching Professionals

The first goal of this phase was to understand the videography rules followed by professionals for lecture video recording/switching, especially to find out simple rules that could be taken over by automatic decision engines on computers. We also asked the professionals to manually edit the 15-minute presentation, which were used for comparison study in later phases. Finally, we played the automatically switched video output and asked the switching professionals for their comments on it.

We talked to three very experienced video switching professionals. Andrew James Macgregor from Department of Computer Science of University of Illinois at Urbana-Champaign has over 10 years of experiences in video editing and production in general, and he has been the chief media supporting staff in UIUC for 8 years, working on lecture recording and live broadcast for both single room and distributed lectures. Paul Riismandel from Liberal Arts has over 15 years of camera switching experiences in capture and live broadcast of distributed classes and events with multi-camera crew. Jeff Carpenter from NCSA at UIUC also has over 10 years of experiences in camera switching, and he regularly supports class capturing and video conferencing needs in NCSA.

8.4.35.1 Phase 1.Step 1 – Introduction

In this step, we first introduced the problem of visual stream management to the switching professionals, and explained our solution that aims at automating this process for relatively

simple scenarios where human users can “teach” computers to understand human preference via a set of rules and scores. We also showed a short demo of our current automatic system output to illustrate the idea, and then explained the goals and procedure of this experiment.

8.4.35.2 Phase 1.Step 2 – Discussion of Videography Rules

For this step, we interviewed each of the professionals on the following questions:

(1) What are the general rules that you follow when editing a multi-camera video session, such as in terms of timing of switching, the layout of video windows, etc.?

(2) What are some of the specific scenarios that entail special rules, such as relative importance on different visual streams during a monologue of the lecturer vs. a Q&A conversation?

(3) What are some of the difficulties involved in day-to-day video switching practice?

(4) What are some of the advantages of human editors you think are not easily expressed in simple rule-like knowledge?

Below is a list of the main points resulting from the discussions.

(1) If the data stream (PowerPoint slides or blackboard writing or transparency writing) is being updated very often, then we should not switch off it too often to distract students from reading it;

(2) It is preferred that the data stream maintains the same size to facilitate students’ reading;

(3) For large classrooms, it is often very hard to find the speaking student, so unless it is an on-going conversation, do not switch to a video stream of a student on a short question;

(4) Sometimes some student does not want to be captured in the video, e.g. for cultural reasons, and the editor needs be flexible to accommodate for that;

(5) Screen layout allows for multiple visual streams to be selected, but also greatly increases the complexity of real-time video switching because of the exponentially growing number of choices;

(6) Human editors all have habits and biases that may affect the quality and predictability of screen output;

(7) Human editor may not be able to continuously pay full attention for a long time, so it is likely that some semantic events may not be captured in time;

(8) Human editors can predict which streams will become interesting and conduct stream switching accordingly;

(9) Showing the face of lecturer and students is almost always important;

(10) Certain scenes should be avoided, such as yawning or chatting.

In summary, there are several important conclusions we can make about the manual visual stream management process as done by video professionals. First, *customizability* is very important since even for a simple lecture room recording scenario there can be a lot of parameters that are changing across different sessions, such as number of visual streams, relative importance of these streams, preferred screen layout, etc. Second, *computer automation* is especially good for repeating and so "boring" tasks that may not be carried out well rigorously by human editors. Third, *automation* is also suitable for scenarios that may be complicated for human editors to deal with but can be dealt with by computers based on simple rules, such as the case for choosing multiple visual streams for screen output in realtime. Forth,

there are *advantages of human editors* that computers can not (or even never) compete with, such as predicting a coming camera switch, switching based on the content of the lecture and words of the lecturer, etc.

8.4.35.3 Phase 1.Step 3 – Manual Switching

Note that since the quality of screen output of a distributed class is very subjective, it is hard or improper to define a "ground truth" for quality evaluation. Therefore, our methodology is to invite the video switching professionals to conduct the switching manually and check how much of the switching decisions made by them can be covered by the automatic switcher.

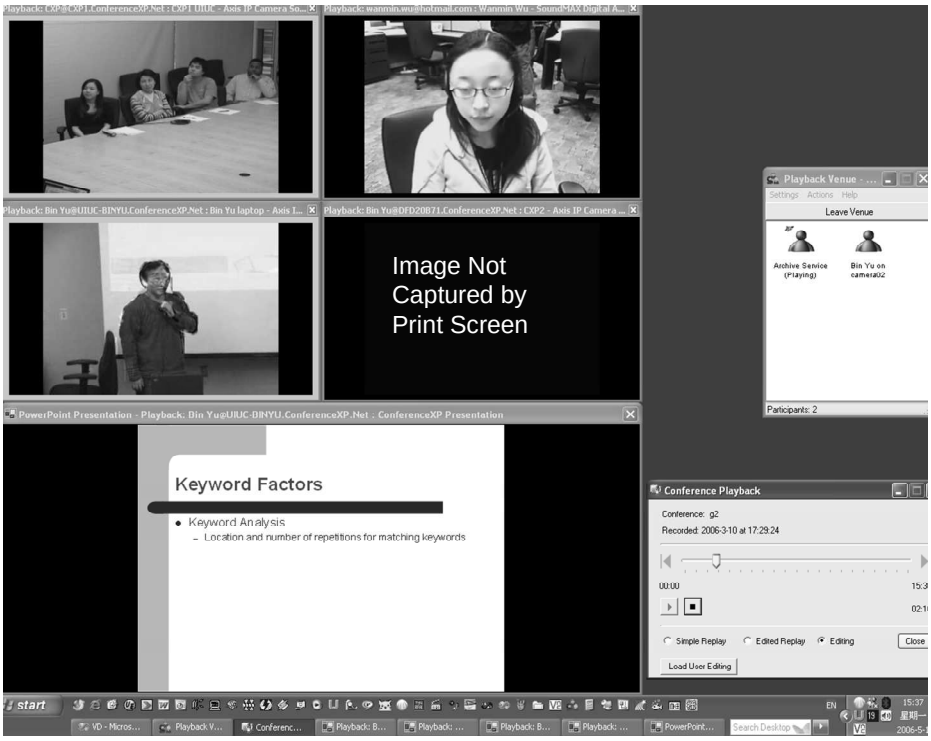

Figure 36. User Interface for Manually Making Stream Selection Choices On-line

We have developed a separate user interface for this experiment, which allows the video switching professionals to watch the playback of all the visual streams of the lecture and make choices on the fly on which video(s) should be selected for screen output to simulate a live

lecture broadcast scenario. A screenshot of the on-line switching interface is shown in Figure 36 above.

The four video streams from the four cameras (one camera image is not shown because of the limitation of “print screen” function on Windows platform) and the PowerPoint slide stream are all shown in tiled screen layout. We assume the PowerPoint stream will always be selected for display to simplify the switching process, and for the 4 video streams the user can select which stream(s) to display by mouse: if he left clicks on a video window, only that video stream (plus the PowerPoint stream) will be selected for display; if he right clicks on a video window, that stream will toggle between being selected or not. On the interface, when a video window is in the “selected” state, its border will be in light green. For example, if at the beginning, the user left clicks video window 1, then the system will log that this user wants both the PowerPoint stream and video stream 1 to be displayed. If the user then left clicks video window 2, then the system will log that the PowerPoint stream and video stream 2 are selected for display. If the user then right clicks on video window 3, then video stream 2 and 3 and the PowerPoint stream will be selected.

We also experimented with another function of “locking a video window as selected”. It works by holding the control key in the keyboard and left click a video window. Clicking an unlocked window will lock it, and clicking on it again will unlock it. On the user interface a locked window has a light blue border. However, from our experiences, out of the 11 users (3 professionals and 8 ordinary subjects to be discussed later), only 2 of them choose to use this function. We assume the reason is that the two simple clicking functions are already good enough for most of the cases, and they are easier to learn in the short time of the user study.

8.4.36 Phase 2 – Comparison Study

In this phase, we wanted to understand to what extend can automation simulate human video switching to manage the visual stream selection and screen layout. Specifically, we conducted the user study by asking the subjects to watch the recorded presentation twice: one time with the virtual director loading the log of the video switching done by one of the video switching professionals and controls the stream selection and screen layout accordingly (***manual***); for the other time, the virtual director automatically controlled which streams were displayed and their screen layout, and the configuration parameters of the virtual director were preset (**automatic**).

We did not tell the subject which method was used for each playback. After the playback, we asked the following question.

"Which playback (first or second) is automatically controlled? Please tell us why in details."

8 human subjects were invited for this experiment, 4 females and 4 male, all in the age group between 20 and 40, and all were average computer users. The results are shown in Table 8-2.

	First playback is automatic	Judgment
Subject 1	**True**	**Correct**
Subject 2	**False**	**Incorrect**
Subject 3	**True**	**Correct**
Subject 4	**False**	**Correct**
Subject 5	**True**	**Incorrect**
Subject 6	**False**	**Incorrect**
Subject 7	**True**	**Correct**
Subject 8	**False**	**Incorrect**

Table 8-2 Judgment by subjects on which playback is automatic

From the table we can see that out of these 8 subjects, only 4 could correctly tell which playback was automatic. Since we have randomly chosen which playback method was used first, we have eliminated the possibility that the playback order would affect the subject's judgment. From the user feedback, even the 4 subjects who were correct were not sure about their answers when they make the selection. Below are some of the reasons given by the subjects in supporting their judgment along with some of our interpretations and comments:

1. Reasons for correctly recognizing the automatic playback as automatically switched

a) *"Not very flexible in terms of timing. Sometimes I can sense a fixed length of time before switching."*

i. This is because some times when no interesting events occur, the automatic switching will be guided mainly by timing, so it becomes visible to a careful observer.

ii. This problem can be solved by adding more randomness in the switching process.

b) *"There are sometimes delays in switching to the speaker."*

i. This is because the automatic switcher follows the minimum waiting time before switching out of the current stream being displayed. Also the voice detection module is programmed in the way that it will wait to accumulate at least 2 seconds audio data before it signals a speaking event, which is necessary to distinguish human speech from noises.

ii. This problem reflects that the quality of the automatic switcher relies heavily on the quality of the underlying semantic event detection. To reduce the switching

delay, more advanced voice detection techniques can be applied, especially if the audio channel is analyzed in combination with the camera video stream.

c) "Sometimes no person or non-front view of lecturer are shown in the video"

i. This is because in our experiment we did not support the functionality of face detection or motion detection.

ii. This problem is similar to the last problem. These pretty simple tasks for human switchers are very hard for computers to accomplish in a reliable and efficient way.

2. Reasons for incorrectly taking the automatic playback as manually switched

a) "I think this playback is pretty smooth"

Several subjects have a positive feedback on the automatic switching result.

b) "Sometimes the video of the audience is switched on before they start to talk, which might be the result of the prediction by the human editor"

i. This is only because we have a maximum timer that specifies how long at most to stay at a video stream before switching off.

ii. Since human switchers often switch off a speaker to show the audiences' feedback or to add some variety to the screen output, the automatic system has a good chance of simulating the human switcher by combining maximum timer with randomness,

3. Reasons for correctly recognizing the manual playback as manually switched:

a) "Sometimes the video of the audience is switched on before they start to talk, which might be the result of the prediction by the human editor."

Without doubt, human switchers have the inherent advantages of understanding the content of the presentation and what the lecturer is referring to, which can be used in the switching process.

b) "It feels less mechanical than the other playback"

i. The other advantage of human switchers is they can make switching decisions based on the "pace of the presentation". If they see that the lecturer does not have much motion or gestures, they will try to make switching more often to make it more interesting to watch; otherwise, if the lecturer is pretty energetic and dynamic, the switcher will make less switching.

ii. We argue that this can be supported by customizing the automatic system. The system parameters for minimum and maximum switching time can be configured for each class and each lecturer by the system administrator in advance, so that the automatic switcher will also exhibit different switching behaviors for different types of lectures.

4. Reasons for incorrectly taking the manual playback as automatically switched:

a) "Sometimes one video is selected for too short before it is switched off".

This is because human switchers do not always keep the exact time since last video switching, so when something interesting shows up in an off-screen video, the human

switchers sometimes tend to switch to it immediately without considering the short switching intervals.

b) "Sometimes the video switches to a student who does not speak"

i. This may be because human switchers' predictions can turn out to be wrong. Unless an event is well orchestrated in advance, unexpected changes may occur. This puts a limit on the amount of predictions a human switcher would make to avoid the risk of wrong prediction, which leads to a smaller gap between human and automatic switching results.

8.4.37 Phase 3 – Manual Switching Experiment

In this phase, we let the subject use the same interface as the video professionals to pick which streams out of the four video streams they would want to see as an audience. We first demonstrated how to use the system using left and right click, and also let the subject see the playback result of the switching. After the subject was familiar with the interface, we asked him to edit a 10 minute lecture video. Then we asked the subject the following questions:

1. "Besides the fact that you may not be familiar with the video authoring interface, what do you think are some of the difficulties involved in realtime switching"

2. "Now let us replay your switching. Please explain what are the criteria that you used when you make the decision of selecting or deselecting a particular video stream?"

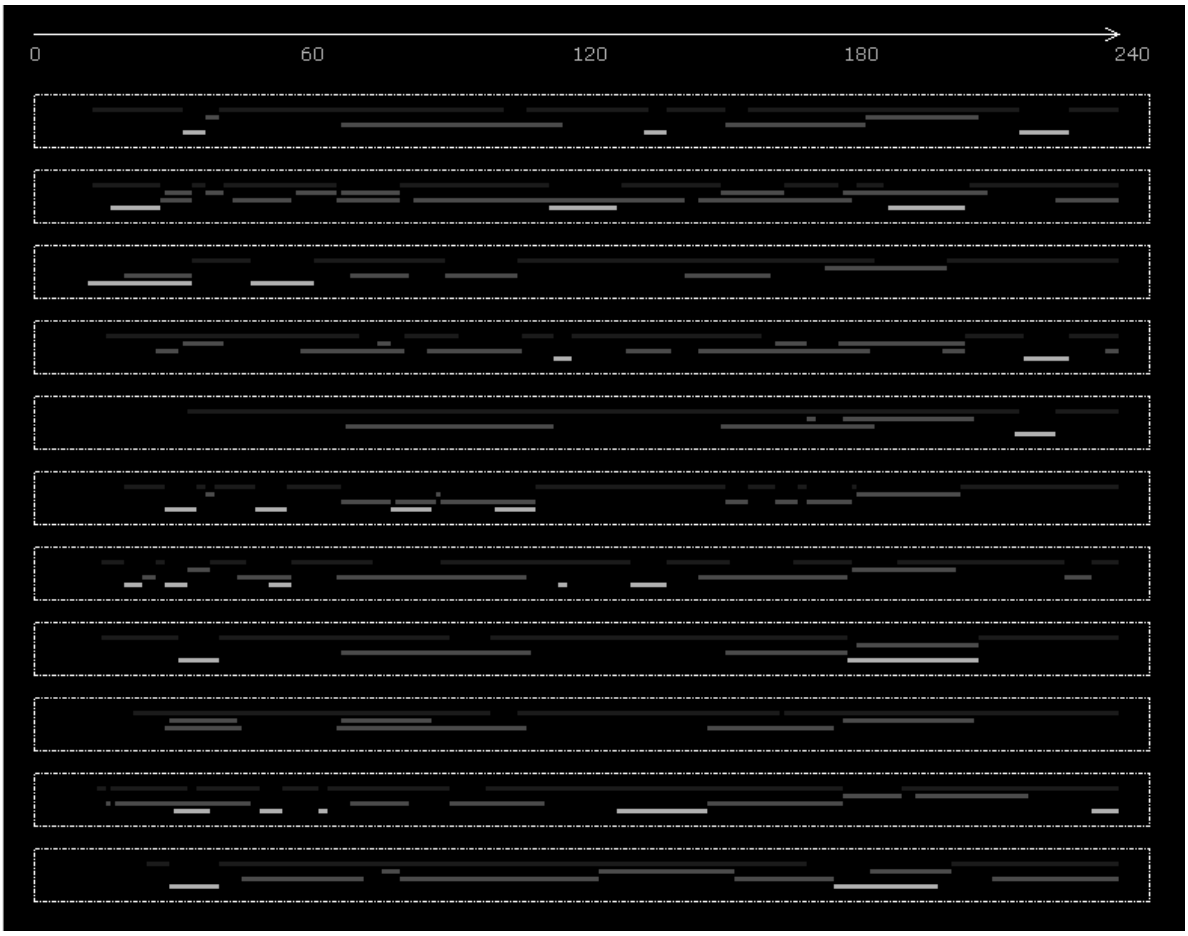

Figure 37. Switching Log of Human Subjects (Including Switching Professionals)

Aaa We have collected 8 switching logs from the 8 subjects, and they are drawn in Figure 37 together with the 3 switching logs from the video switching professionals. At the top of the Figure is the time axis representing the data of 240 seconds lecture starting from the beginning of the presentation, and the 11 (3 for the video switching professionals at the top and 8 for the focus group at the bottom) logs are drawn one by one vertically. Each log has a white dotted box surrounding it, and the four streams are represented by line segments of four colors (red, blue, green and yellow from top to bottom). For each time point (a one second segment on a time line), if a stream is selected by the user for display, then that time-point is drawn in that stream's corresponding color. It is obvious from this figure that there is hardly any obvious

pattern in the switching logs, which reflects the diversity and randomness in human preferences in general. To find a "consensus" among majority of the human subjects, we processed the logs in the following way. We define a parameter called *Overlapping Ratio*, which measures the percentage of human subjects who would agree on selection of each time point. For example, if Overlapping Ratio is *0.8*, then for each time point *t* and each of the four streams, we examine each switching log to check whether that subject has selected that time point to be displayed. If more than *80%* of the switching logs say that point of that stream is selected, then we count that point in a final *consensus selection stream*. By changing the Overlapping Ratio, we can get different consensus selection streams. As shown in Figure 38 below. From the figure we can see that even at a high Overlapping Ratio of 1.0 (meaning that everyone has to agree on the selection), we can still get a consensus stream that almost covers the who 240-second timeline. As the ratio decreases, more time points are found to be acceptable by the majority of the subjects.

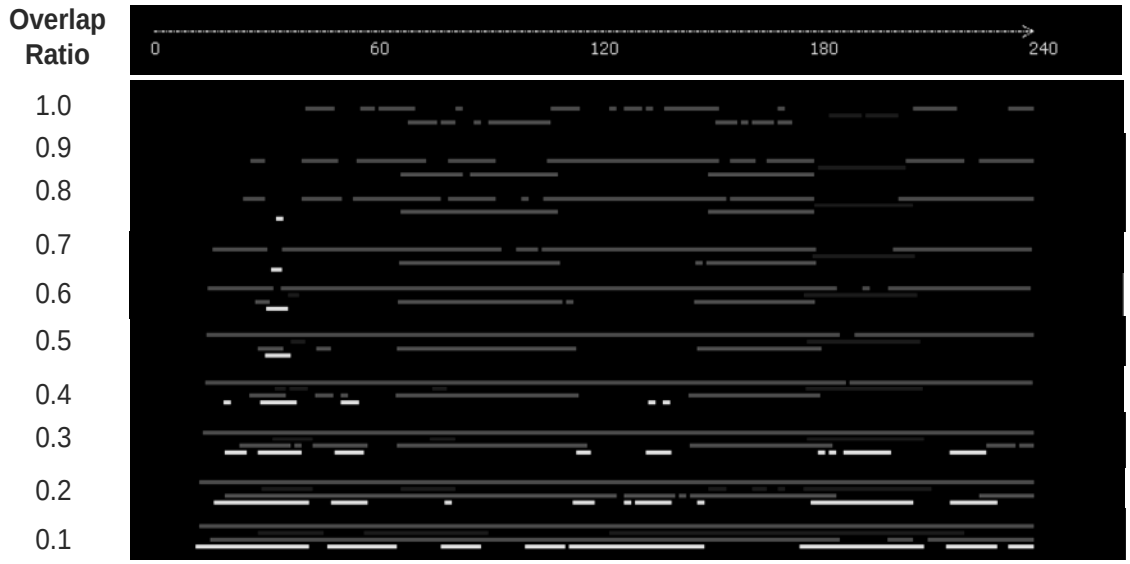

Figure 38. Consensus selection streams corresponding to varying Overlapping Ratio

With the knowledge of the consensus among the majority of the human subjects, we try to test how much the automatic switching director can meet with such human preferences. As in a real lecture scenario, the user (system administrator, video support staff, etc.) of MyView system would configure the parameters (scores, rules, timing constraints) according to the lecture content and lecturer and students' preferences, we followed a similar process in this experiment. Specifically, we configured the parameters of the automatic virtual director as shown in Table 8-3 below.

	Lec Frt View		Lec Side View		Room Aud		Remote Aud		PPT Slide	
	Silent	**Talking**	**Silent**	**Talking**	**Silent**	**Talking**	**Silent**	**Talking**	**Stale**	**New**
Score	1	2	0	0	1	2	1	2	3	3
Min Time	6	10	6	8	6	8	6	8	10	10
Max Time	10	30	10	30	10	30	10	30	30	30

Table 8-3. Parameter Configuration For Automatic Virtual Director To Meet Consensus Stream

From the table we can see that the parameter setting is in fact pretty straightforward. For three of the mostly selected video streams, we only need to set their "silent" view to be score 1 ("show if room") and their "talking" view to be score 2 ("must show with low priority") so that the talking views are captured. We also set all views of the PPT slide stream to be 3 ("must show with high priority") so that it is always selected and stays on the top part of the screen area. The minimum time and maximum time for switching are also from common sense judgment. With such a setting, we get an automatic switching log as shown in Figure 39 below.

Figure 39. Switching log from the automatic switching conducted by the virtual director

We can see that this result resembles the consensus selection stream relatively well. Statistically, the average number of line segments (i.e. the number of switchings) for all the switching logs is 29.18, and the number of line segments in this automatic switching log is 29. The average length of each line segment for all the switching logs is 11.21, while the average length for the automatic switching log is 11.69. Furthermore, we calculated what percentage of the consensus selection stream can be captured by the automatic switcher according to different Overlapping Ratios, and the result is shown in Figure 40.

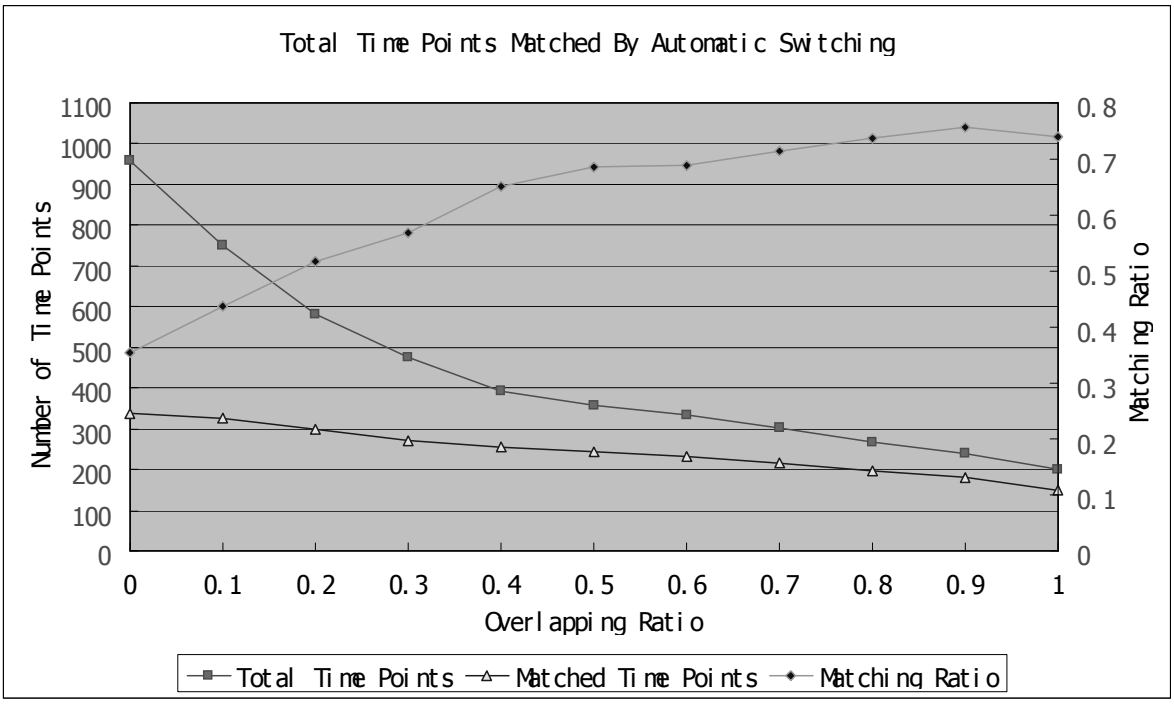

Figure 40. How Much of The Consensus Selection Stream Can Be Captured By Automatic Switching

In the figure, the X axis is increasing Overlapping Ratio (which means stronger consensus among all subjects), the left Y axis is the number of time points of all four streams, and the

right Y axis is the Matching Ratio – How many of the time points in a consensus stream are selected by the automatic switcher. The higher downward sloping curve is the total number of time points in the consensus stream for each Overlapping Ratio, while the lower downward sloping curve shows the number of time points selected by the automatic switcher. The upward sloping curve is the percentage between these two numbers, which represents how well the automatic switcher could capture the time points in the consensus selection stream. As we can see, at an Overlapping Ratio of 0.5 or higher, the automatic switcher can already capture about 70% of the consensus. Considering the diversity and randomness of human selection itself, this is a pretty good proof that the automatic switcher has managed to tailor to the preference of the majority of the human subjects.

8.4.38 Phase 4 – Feedback and Comments

In our questionnaire, we asked the human subjects to write down their feedback and general comments of our system. Some comments are listed below:

- "It is amazing to see the automatic switching result"
- "It is not easy to distinguish between the two switching methods"
- "It is not easy to respond to real time events and make switching decision as a human switcher"

8.4.39 Conclusion of User Study

We find that both manual switching and automatic switching have their own advantages and own drawbacks. For manual switching, the cost is normally higher, and the switching results are generally affected by the personal preference, habit, attention limitation and prediction

errors of the human switcher. On the other hand, for automatic switching, the results are affected by the variety and accuracy of automatic semantic event detection, the lack of predication, the mechanical aspect of switching, and the complexity of rules and parameters to be specified. We have also found a high variance in human preferences as to what should be selected and what not, which makes it even harder for automatic switcher to satisfy human viewers. Nevertheless, the core set of consensus stream segments (over 80% for a half-majority vote) can be captured by the automatic method with a straightforward parameter configuration. Overall, automatic switching systems can produce high quality screen outputs comparable with that of human switchers, and they can be easily customized to meet different human preferences.

8.5 Conclusion

We have already conducted simulation experiments to verify the stream selection and screen layout calculation algorithms, and the results prove the functional correctness and efficiency of our solution. We have also presented the design and implementation of the MyView presentation recording/broadcasting system. Results from the user study have demonstrated the usability of our solution.

Chapter 9. RELATED WORK

In this Chapter, we will first discuss existing solutions for visual space management problem, and then give an overview of related work in other research areas.

9.1 Existing Solutions for Visual Space Management

The problem of visual space management has manifested itself in many different applications, and we will classify existing solutions according to the main approach they adopt in solving the problem.

9.1.40 Show All Visual Streams

Systems in this category (e.g. [17] [19]) adopt the simple and straightforward approach of presenting all visual streams to the user. For example, with the distributed learning system Virtual Auditorium [17], video image of all remote students are displayed on a large display wall in a tiling screen layout. The user (viewer) does not change the layout of these video windows on the screen, but chooses which visual stream to watch and ignores the other videos.

One of the key advantages of such a “show all” solution is that the users have access to all the available visual streams related to the current application at any time. Therefore, the users have the maximal freedom to choose which visual stream to look at, and it is very easy for the user to switch his visual attention to another visual stream. The other key advantage is its simplicity: it is straightforward to implement such a solution, and the users can easily understand how all the visual streams are managed on the screen.

Nevertheless, there are some inevitable limitations of such a “show all” approach. First, in a real world application, the total screen resolution required by all visual streams to be shown at normal quality (not to mention high visual quality) can easily exceed the total amount of available screen space (determined by the available display device and the distance between the user and the display device). Therefore, oftentimes the visual streams will have to be scaled down and squeezed into the limited screen space, which fails to provide the desirable visual quality that is essential to viewer experience, and also leads to scalability issues since the total number of visual streams is limited. Second, since at any given time point the user would be focusing on a single visual stream, the presence of all the other streams on the screen not only wastes valuable screen space, but also distracts the user from watching the stream he is interested in (remember that all the visual streams may be continuously changing). Third, while the user is focusing on one visual stream, he would have to rely on his/her peripheral vision to discover interesting events in other visual streams. This is not a reliable solution, and the problem of “missing event” may occur. For example, suppose a soccer game is captured by 30 cameras. If a viewer is most interested in the camera showing the frontal view of a particular player, it would be extremely hard for him to browse through all the 30 video streams and find what he wants. Finally, from a social perspective, in many multi-stream applications (such as in conferencing applications), a participant may feel uncomfortable to know that he/she is always shown on screen for all remote participants even when he/she is not speaking or actively involved in the conference.

9.1.41 Viewer Selection

One step forward from the “show all” approach, many systems (e.g. ConferenceXP [4], OpenTV [8], Microsoft IP TV [5]) adopt a “viewer selection” approach, where all the visual streams are presented to the user, and the user chooses which one to watch in high resolution (in large video window or in full screen mode). For example, with the ConferenceXP Research Platform [4], the display windows of all the visual streams, such as cameras videos, the PowerPoint presentation, WWW browsing, whiteboard, etc., are tiled one by one on the screen on initialization. Then the user can manually arrange these windows on the screen, such as resizing a window or placing it at a different location on the screen, and also minimizing/maximizing a particular window. For another example, many leading Interactive TV (iTV) service providers, such as OpenTV [8] and Microsoft TV [5], support multiple view angels for sports game broadcast: all view angle choices are presented in split-screen mode by default, and the viewer can pick any one video to watch in full screen. In the Gaia system [37], Roman proposed an application framework where the content presentation is decoupled from the content modeler and controller, and the viewer can assign what content is to be shown on which display devices using a script language.

One of the key advantages of such a “viewer selection” approach is that high visual quality can be achieved. No matter how many visual streams are involved in a session, the user can always choose to allocate the whole screen to a single visual stream that interests him/her most at a time. In addition, the user has complete freedom in customizing how the visual stream windows are arranged on the screen, so each user can tailor the screen output on his/her display device based on personal preference.

However, such an approach also has two obvious limitations. First, as the content of different visual streams keeps changing over time, the user may have to frequently manually adjust the screen layout of all visual stream windows, which would distract him/her from the main task (e.g. enjoying a show or collaborating with remote participants). Second, the problem of "missing event" is aggregated, since when the user allocates all the screen space for one specific visual stream, he/she does not have a clue about what is happening in other streams, so it is very likely that he/she will miss some interesting shots that he would like to watch.

9.1.42 Professional Human Switching

To further improve the quality of visual stream selection and rendering, a widely adopted approach is to hire human operators to "direct" the stream selection and rendering process. For example, in the ePresence system [13] for distant learning, a "human moderator" is hired to manually select a particular video stream to send to remote participants. Also, in the TV/film industry, professional human editors will decide which camera video to put on air for the audiences.

Such an approach has two key advantages. First, the users are saved from any distractive operations, so that they can focus on their main task. Second, much higher production quality is preserved: not only because the visual quality is higher, but also because the human editor can apply many videography rules in the directing process. For example, he may limit the minimum and maximum time for a particular camera view to be shown, and he may follow particular "idioms" [12] to deliver the visual message to the audiences. For instance, as documented in [12], when filming a scene of two person conversation, it is best to show an

overview shot of the two persons' faces, followed by alternating views of each person's face and the overview shot.

Although such an approach has been well accepted by both the filming industry and audiences, it still has some drawbacks. First, the editing crew is often not affordable in many applications. For example, as distant learning becomes widely used, we cannot expect one human editor to sit in each classroom to conduct video editing. Second, the human editor makes the judgment of which visual stream to show on air based on both what he deems important and what he assumes is interesting to the audiences. This may be different from the audiences' actual judgment and interest. Considering that different audiences may have different interest, the human editor can never satisfy everyone so long as he is producing a single video program that is presented to all audiences. Third, the human editor is still a human being, and he still needs to watch all visual streams to find the one that is interesting. Although he may have been trained to be more proficient in predicting and discovering interesting shots, he may still miss some interest moments that could have been presented to the audiences.

9.1.43 Automatic Stream Switching

To avoid the cost of human operators, several systems (e.g. [23] [37] [34] [2]) have proposed to use an automatic stream switching or "virtual directing" approach: to have the computer automatically select which visual stream(s) to render on the screen. These systems can be further classified into the following two categories based on how the automatic switching decision is made.

9.1.43.1 Automata Based Automatic Switching

Example systems in this category are iCam [38] and AutoAuditorium [2]. In the lecture capturing system iCam [38], Rui et al. propose to switch between the audience camera and lecturer camera based on a finite state machine. The states of the automaton represents the current views of each camera and which camera is selected to put on air, while the transitions between the states represent various events such as detection of audience speaker or timer expiration. Similarly, in AutoAuditorium [2], a predefined automaton is used to direct the switching between the lecturer camera, the presentation slide and the global view of front stage of the lecture room.

There are three key advantages of such automaton based systems. First, the cost of human editor is saved, which greatly decreases the total cost of deployment of such systems. Second, high visual quality is preserved since the screen space is often allocated to one or a few visual streams. Third, the users are not distracted from their main tasks because the system works on its own.

However, there are still several aspects with such automata-based automatic switching approaches that can be further improved. First, the interest model for viewers is often simple and predefined. For example, it is generally assumed that when some person begins to talk, the camera view of that person's face should be switched on screen as soon as possible. However, a viewer may want to stick to other views, or he may be more interested in watching the reaction of others rather than the speaking person. A fixed and predefined interest model prevents customization based on the preference of a particular user, and it does not support more complex user interest variations in real-world applications. Second, the finite state

automaton needs to be manually authored by the system administrator, typically in some scripting language, which makes it very difficult for the system administrator or the viewers to modify the default stream selection rules. For example, iCam's automaton is specified as a cascade of decision rules, which are interdependent to each other. To modify the system to support one more camera, or to support a camera with different states and visual events from existing cameras, the whole automaton needs to be rewritten. This whole process is tedious and error prone.

There is another category of work where virtual camera view points are intelligently controlled, such as the Intelligent Camera Control [19] work by Steven Drucker. In the intelligent camera control work, low level measurements are made on the 3D virtual objects, such as the visibility and distance, which are then used to calculate the best virtual camera viewpoint. However, the user's preference is not considered, so the output is not easily customizable as in our visual stream management framework.

9.1.43.2 Score Based Automatic Switching

The other category of automatic systems (e.g. [23] [34]) switch between different visual streams based on their *scores*. The score generally represents a measurement of how much a viewer desires to watch a particular visual stream, and the score for each stream will change over time based on the change of information delivered in each visual stream and also the change in user interest. By comparing the scores, the system would know which streams are the most desirable. In different systems, the score is calculated in different ways. For example, in MPI-Video [23], interesting objects/people are tracked in multiple camera views to generate

a world model. A “best view” is then computed based on properties of these objects/people, such as “object/people occlusion degree” or “distance between object/people and the camera”, etc. On the other hand, in [33], multiple cameras monitor the same room from different angles, and the “best view” is selected based on how clearly the face of the human subject is captured from each camera view.

Besides the three key advantages described above for automatic stream switching solutions, there is another unique strongpoint for such score-based systems: the user is saved from authoring the complicated finite state automata, since the system does not maintain any state information and only relies on the current scores of all visual streams to make rendering decisions.

However, this score-based approach shares the same limitation with the automata based approach in that the automatic rendering output is uniformly defined for all users, so it is not possible to customize the output based on each user’s preference. In addition, the lack of state information (which visual streams are showing what, which visual streams have been selected for how long) limits the rendering capability of systems. For example, it is not possible to switch from a high scoring stream to a low scoring one, and it is not possible to switch to a particular stream based on the current rendering state of the system according to some predefined idiom.

Overall, many researchers have explored the solution space for visual space management in different directions. Although they have achieved great success in building solid systems that are widely used, they still fall short of the goal of providing high quality, interest driven,

customized, automatic visual stream selection and rendering. This is the driving motivation behind our work, as will be introduced in Section 1.3.

9.2 Existing Work in Related Areas

9.2.44 Video Highlight Generation

The question of "which video is more interesting" as asked in the visual space management problem is similar to the question of "which parts of a video is more interesting", and the latter is a classic problem studied in the area of video abstraction, especially for generation of video highlights [28]. A video highlight is a short video skim consisting of the most interesting segments of a video clip. Most highlight generation solutions follow a three step approach:

1. Detect *video shots* based on some low level video features, such as continuous camera motions or color information, etc., or high level semantic video features, such as closed caption or known structure in the video;

2. Evaluate each video shot for its relevance or interest level and generate an *interest score*;

3. Concatenate top scoring video shots into a short video clip based on the desired length of the highlight.

The key challenge lies in the second step, i.e. how to define a good model for viewer's interest in each video shot. Many solutions have been proposed, generally falling into the following two categories: complete automatic solutions and semi-automatic solutions based on user interaction.

One example of complete automatic highlight generation is the VAbstract system [35] that automatically produces movie trailers. In this work, Pfeiffer et al. specify that a "good quality

abstract" should contain *important objects/people* and *events* (such as dialog and scenes with strong motion). In [16], Chang et al. use a set of predefined *events* (such as "home run" and "within diamond play") to evaluate baseball video shots, while seven types of scene shots (such as "pitch view" and "catch close-up") are used to train a Hidden Markov Model that detects those events. In [31], Ma et al. propose a user attention model that evaluates each shot using attention attraction indicators, such as high motion, strong intensity contrast, face of people, etc. Overall, such complete automatic approaches generate video highlights that capture the important objects/people and high motion/contrast scenes.

On the other hand, some semi-automatic solutions utilize the viewers' interactions as a feedback to the relevance/interest evaluation process. For example, in MediaMiner [41], Syeda-Mahmood et al. learns a Hidden Markov Model for viewer interest as the viewers browses video clips using a traditional video player. In VideoAbstract [44], Toklu et al. propose a hybrid approach that automatically generates a video storyboard, and then utilizes user input to refine the storyboard into a meaningful video summary. In one of our previous work ([49]), the user's video browsing log is analyzed using link-analysis algorithms for web document retrieval to derive interest scores for video shots. Overall, such automatic systems gain more semantic understandings and higher level of user customization at the cost of requesting more user interactions.

9.2.45 Visual Event Detection

Visual event detection has been an active and challenging research topic, and many solid results have been achieved in recent years to support visual event based systems.

Many face detection and tracking algorithms have been proposed [22][45] with promising results. For example, in [39], Schneiderman proposes a feature based cascaded object detection algorithm that detects both frontal and non-frontal face robustly. In [32] high precision face tracking is achieved using Discriminator technique. The problem of talking face detection has been studied in [18] and [27], and over 75% detection rate has been achieved by correlating audio and visual information. In [26], Kawato et al. focus on the point “between-eyes” to detect head nodding/shaking with very high precision. In [43], Teo et al. present a facial expression recognition system that recognizes expressions like anger, happiness, sad and surprise with over 70% accuracy. Hand gesture recognition precision has also been improved to over 95% in [24] and [30].

Chapter 10. CONCLUSION AND FUTURE WORK

10.1 Conclusion

The problem of visual space management can be found in more and more Multimedia applications where multiple correlated visual streams are presented to the user on a limited display space. In this work, the main contributions are as follows:

- We present a novel model for visual space management. We adopt a shot-based model that captures the key characteristics of a visual stream's changing visual information, and a generic score-based user interest model that helps the system to predict user interest. Our model is not limited to a specific type of visual stream, so it can be applied to a wide range of Multimedia applications.
- We develop a 3-phase visual space management framework that combines modeling of visual stream source and the selection/rendering of visual streams. The stream selection algorithm is unique in the sense that it automatically translates the high level score based user interest model into the timed automaton that serves as the decision engine for stream selection process, thus solving the problem of complexity and scalability issues of previous solutions that rely on manually authorized automata. The screen layout calculation algorithm addresses the problem of presenting multiple visual streams on multiple display devices or screen regions for the first time.

- We validate the model and framework via a pilot system called MyView based on our solution. We evaluate our solution through simulation experiments as well as a user study to prove the usability of our system.

10.2 **Future Work**

There are a lot of potential areas that can be explored from the current stage of the work.

First, currently we only focus on the screen management problem, while the semantic knowledge as to which stream is more interesting can be further utilized in many other applications. For example, one direction is to utilize this knowledge to save the streaming bitrate by only transmit those streams that are of higher scores and to be selected. The knowledge can also be used to generate a short video summary of the original full raw video.

Second, we have been working with the relatively simple scenario of distributed lecture presentation, while the idea of our solution can be further developed to other more complicated scenarios. For example, one interesting application is the view selection in Teleimmersion environment, where any virtual view point can be selected for displaying the 3D pixel model of the target scene/object. In this setting there are theoretically innumerous number of virtual view points (and so visual streams) to choose from, and our solution can be extended to automatically select a best view angle. Another application is live sports broadcast, where the need for customization is strong, and there can be many interesting visual events that can be automatically detected.

Third, for the distributed lecture presentation scenario, the audio/video system needs to be further improved to render a superior user experience in combination of our automatic video switching system. Specifically, microphone arrays can be adopted to detect the location of the

speaking audience member, and this knowledge can be used by the camera array to zoom into the face of that person to improve the visual quality of distributed interactions.

APPENDIX A. PSEUDO CODE FOR TIMED AUTOMATON GENERATION IN UPPAAL

A.1 Generation of Shot Type Change and Maximum Timer Triggered Automaton

A.1.1 Generation of States

For each shot type combination of all visual streams $<V_1, V_2, ..V_N>$ {

If (there exists at least one visual stream VS_j, $S[j][V_j] \geqslant 2$) {

// select “must show” stream shot types

Shot Type Combination = $<V_1, V_2.. V_N>$;

Selected Stream Index = indexes of all streams with a shot type of score of 2 or 3;

Minimum Time = empty;

For each selected visual stream *i* with shot type *j* in Selected Stream Index {

Minimum Time = Minimum Time AND “$curtime \geqslant Min[i][j]$”

}

Maximum Time = 0 ; // not used

Output one new state *<Shot Type Combination, Selected Stream Index, Minimum Time, Maximum Time>*

} else if (there exists $K \geqslant 1$ visual stream VS_j, that $S[j][V_j] == 1$) {

// since these streams will be shown in rotation, add one state for the case of

// selecting each of them

For each visual stream VS_j, if $S[j][V_j] == 1$ {

Selected Stream Index = "*j*";

Minimum Time = $Min[j][V_j]$;

If ($K > 1$)

Maximum Time = $Max[j][V_j]$;

Else {

Maximum Time = 0; // not used, no other stream to rotate to

}

Output one new state < <V_1, V_2.. V_N>, Selected Stream Index, Minimum Time, Maximum Time>

}

} else {

// the rare case of all streams are in a shot type with "do not show" score,

// so by default show stream 0

Selected Stream Index = "*0*";

Minimum Time = $Min[0][V_0]$;

Maximum Time = 0; // not used

Output one new state < <V_1, V_2.. V_N>, Selected Stream Index, Minimum Time, Maximum Time>

}

}

A.1.2 Generation of Shot Type Change Triggered Transitions

For each ordered pair of two states (S_1 and S_2) with one and only one stream VS_k has different shot types in shot type combination {

Form state index = S_1; To state index = S_2;

Timing constraint = S_1's Minimum time;

Let V_{new} be stream VS_k's shot type index in S_2;

Triggering channel = "ck_V_{new}";

Update = "curtime := 0"

For each visual stream *i* {

If (*i* is in S_2's selected stream index list)

Update = Update + "selected[i] = true";

Else

Update = Update + "selected[i] = false";

}

Output one new transition < *From state index, To state index, Timing constraint, Triggering channel, Update*>

}

A.1.3 Generation of Maximum Timer Triggered Transitions

For each ordered pair of two states (S_1 and S_2) with the same shot type combination {

Form state index = S_1; To state index = S_2;

Let k be the selected stream index in S_1, and assume its shot type in S_1 is V_k

Timing constraint = "curtime $\geqslant$ *Max[k][V_k]*";

Triggering channel = empty;

Update = "curtime := 0"

For each visual stream i {

If (i == k)

Update = Update + "selected[i] = true";

Else

Update = Update + "selected[i] = false";

}

Output one new transition <*From state index, To state index, Timing constraint, Triggering channel, Update*>

}

We will use a simple example to illustrate the automaton generation process. We will assume there are two streams (stream 0 and stream 1), and each stream will have two shot types (shot type 0 and shot type 1). Shot type 0 of both stream is assigned the score level 0, and shot type 1 of both stream is assigned score level 1. Then after the generation of states and transitions based on the algorithm described above, we will get an automaton as shown in Figure 41:

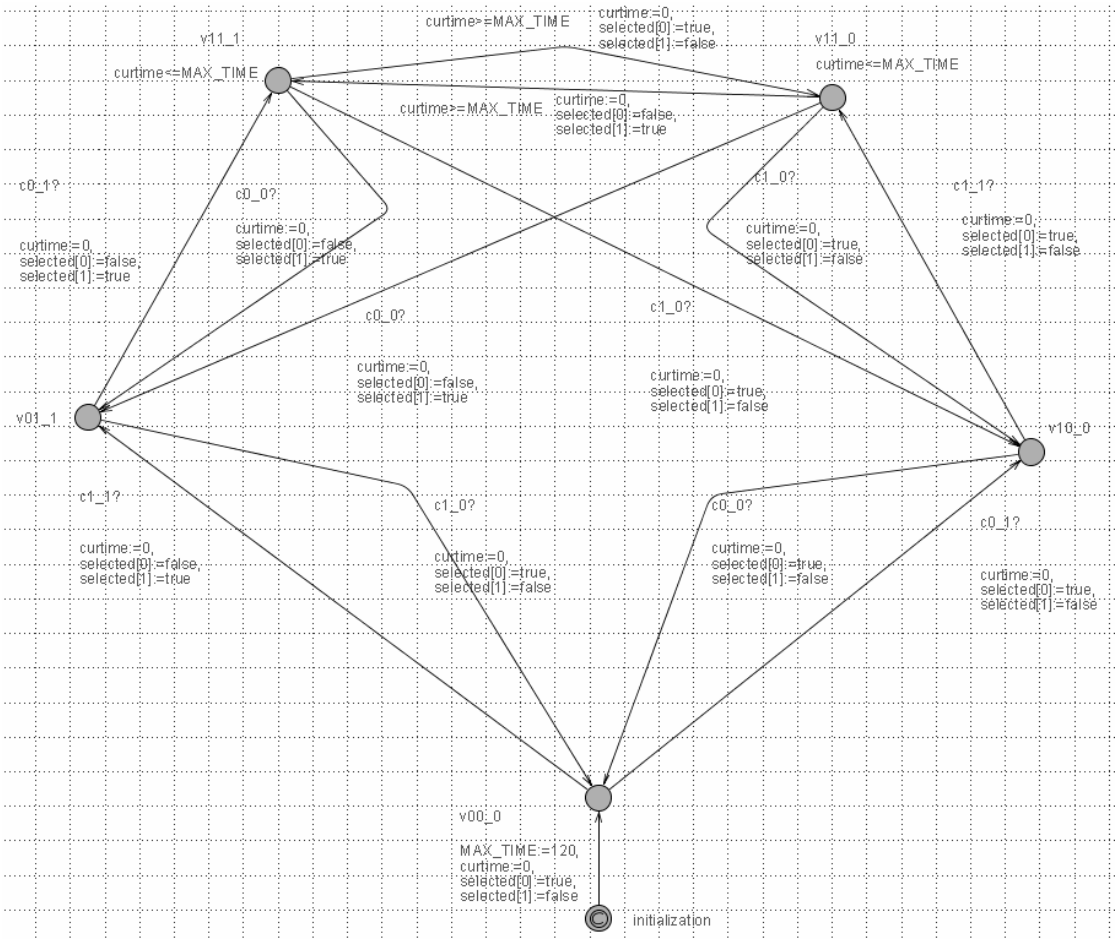

Figure 41. Basic Automata with Transitions Triggered by View Change and Maximum Timer

There are totally 5 states besides the initialization state at the bottom:

- "*V00_0*" represents the state that stream *0* is in shot type *0* (the first *0*), stream *1* is in shot type *0* (the second *0*), and stream *0* is selected (the third *0*).
- "*V01_1*" represents the state that stream *0* is in shot type *0* (the leading *0*), stream *1* is in shot type *1* (the first *1*), and stream *1* is selected (the second *1*).

- "*V10_1*" represents the state that stream *0* is in shot type *1* (the first *1*), stream *1* is in shot type *0* (the middle *0*), and stream *0* is selected (the second *1*).
- "*V11_0*" represents the state that stream *0* is in shot type *1* (the first 1), stream *1* is in shot type *1* (the second *1*), and stream *0* is selected (the *0*).
- "*V11_1*" represents the state that stream *0* is in shot type *1* (the first *1*), stream *1* is in shot type *1* (the second *1*), and stream *1* is selected (the third *1*).

8 of the 10 transitions between the 5 states in the automaton represent state transitions triggered by shot-type change, and they are all marked with the trigger in the form of "*c[stream_index]_[shottype_index]?*". For example, the transition from *V00_0* to *V01_1* is marked with the event trigger channel "*c1_1?*", meaning that the change of shot type of stream *1* (the first *1*) to its shot type *1* (the second *1*) will trigger the state change from *V00_0* to *V01_1*. The other 2 transitions between *V11_1* and *V11_0* represent the case that the two streams are selected in rotation, and these transitions are enabled by the maximum timer. This is enforced by two conditions. First, the invariant of the state *V11_1* and *V11_0* says "curtime $\leq$ MAX_TIME", meaning that the automaton must transit out of these states within MAX_TIME units. On the other hand, the 2 transitions between them says "curtime $\geqslant$ MAX_TIME", meaning that these transitions are enabled only after the automaton has stated in one of the two states for at least MAX_TIME units. Therefore, their combined effect is that if no shot type change occurs, the transition will be fired exactly at MAX_TIME.

A.2 Automaton with Support for Minimum Time Constraint

A.2.1 Generation of Idiom States

For each state *S* in automaton {

Shot Type Combination = *S*'s shot type combination;

Selected Stream Index = *S*'s selected stream index;

Minimum Time = empty; // not used

Maximum Time = Minimum time of *S*;

Output one new state <*Shot Type Combination, Selected Stream Index, Minimum Time, Maximum Time*>

}

A.2.2 Generation of Transitions from a normal state to a hold state

For each ordered pair of two states (S_1 and S_2) with one and only one stream VS_k has different shot types in shot type combination {

Let *T* be the current existing transition from S_1 to S_2;

Let S_3 be the hold state for S_2;

Form state index = S_1; To state index = S_3;

Timing constraint = empty;

For each selected visual stream *i* with shot type *j* in S_1 {

Timing constraint = Timing constraint OR "*curtime < Min[i][j]*"

}

Triggering channel = *T*'s triggering channel;

Update = empty;

Output one new transition <*From state index, To state index, Timing constraint, Triggering channel, Update*>

From state index = S_3; To state index = S_2;

Timing constraint = empty;

For each selected visual stream *i* with shot type *j* in S_1 {

Timing constraint = Timing constraint AND "*curtime* $\geqslant$ *Min[i][j]*"

}

Triggering channel = empty;

Update = T's Update;

Output one new transition <*From state index, To state index, Timing constraint, Triggering channel, Update*>

}

A.2.3 Generation of Transitions from a hold State to another hold State

For each ordered pair of two normal states (S_1 and S_2) with one and only one stream VS_k has different shot types in shot type combination {

Let T be the current existing transition from S_1 to S_2;

Let S_3 be the hold state for S_1; Let S_4 be the hold State for S_2;

Form state index = S_3; To state index = S_4;

Timing constraint = empty;

Triggering channel = T's triggering channel;

Update = empty;

Output one new transition *<From state index, To state index, Timing constraint, Triggering channel, Update>*

}

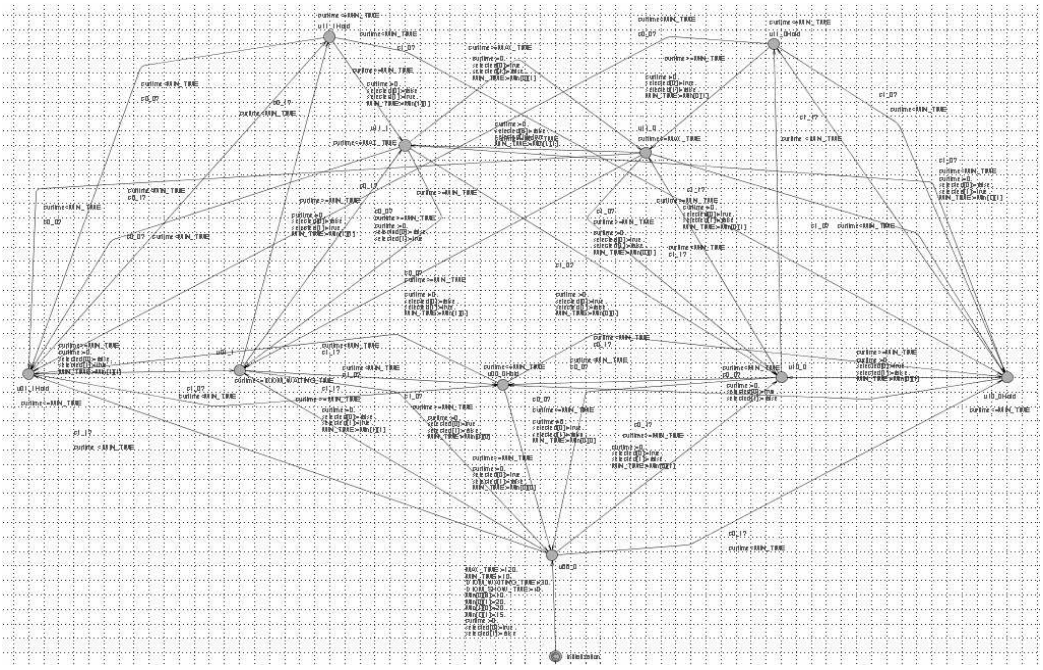

Figure 42. Example Automata With Support For Minimum Time Constraint

Example System

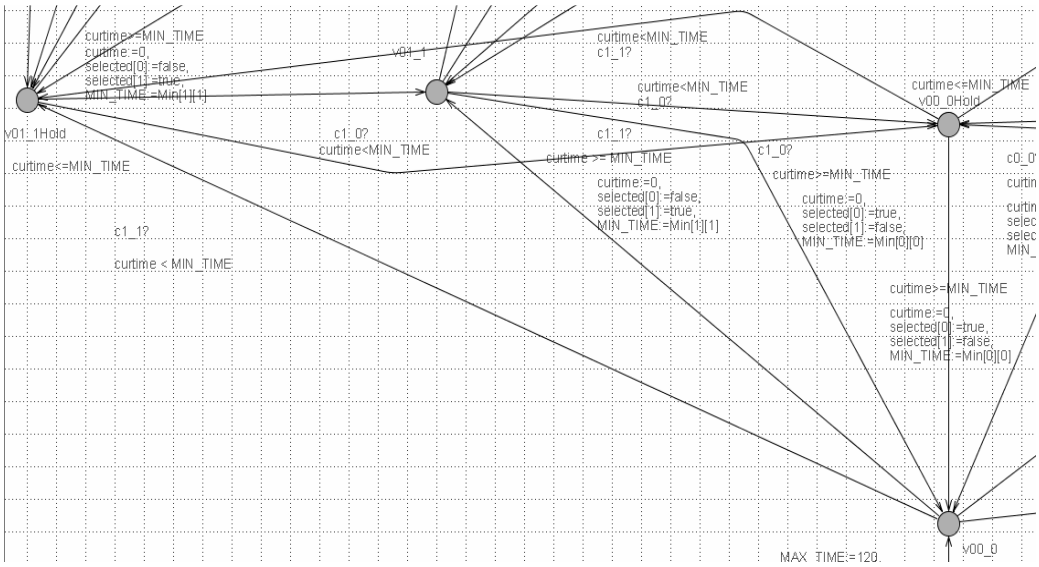

Figure 43. Details of Portion of the Enhanced Automata Shown In Figure 17

We will continue on the example system we presented in the last Section. After adding states and transitions to support minimum time constraint, the new automaton is shown in Figure 42 (overview) and Figure 43 (zoom in view shown below).

In the new automaton, the following changes are made:

- 5 hold states are added corresponding to the 5 original output states: *V00_0Hold, V01_1Hold, V10_0Hold, V11_0Hold, V11_1Hold*. From each of these hold states, one transition is added to the corresponding output state. For example, the transition from *V01_1Hold* to *V01_1* is marked with the condition "*curtime* $\geqslant$ *MIN_TIME*", meaning that

if the automaton is in the hold state and more than MIN_TIME has passed since the last time selection was changed, then the automaton should change to the output state *V01_1*. Note that such transitions will also reset the clock "*curtime := 0*" and MIN_TIME.

- All the original shot type-based transitions are enhanced with a new condition "*curtime* $\geqslant$ *MIN_TIME*", meaning that these transitions will not be enabled when the clock has not passed MIN_TIME.
- New transitions are added from original output states to hold states to represent the case when a shot type change occurs before the MIN_TIME duration is passed. For example, the transition from state *V00_0* to *V01_1Hold* has a guard condition "*curtime* < *MIN_TIME*", meaning that if the curtime clock is less than MIN_TIME, this transition will be fired and the automaton will change to the *V01_1Hold* state.
- New transitions are added between the Hold states to represent the case when a new shot type change event occurs while the automaton is already in a Hold state.

A.3 Automaton with Support for Idiom Based Transitions

Each idiom specifies that if the current selected streams are in the *Idiom_Current_Selection_List* with a specific *Idiom_Shottype_Combinataion* for *Idiom_Waiting_Time*, then change selected streams to *Idiom_New_Selection_List* for *Idiom_Show_Time*. To support such idioms, we need to add the states that represent the new selection state, and also transitions to and from them.

A.3.1 Generation of Idiom States

For each idiom *Idiom*$_x$, for each state *S* in automaton that satisfies *Idiom*$_x$'s *Idiom_Current_Selection_List* and *Idiom_Shottype_Combinataion* {

Shot Type Combination = *S*'s shot type combination;

Selected Stream Index = *Idiom*$_x$'s *Idiom_New_Selection_List*;

Minimum Time = empty;

For each selected visual stream *i* with shot type *j* in Selected Stream Index {

Minimum Time = Minimum Time AND "*curtime* $\geqslant$ *Min[i][j]*"

}

Maximum Time = *Idiom*$_x$'s *Idiom_Show_Time*;

Output one new state <*Shot Type Combination, Selected Stream Index, Minimum Time, Maximum Time*> }

A.3.2 Adding Transitions from normal states to Idiom states

For each Idiom *Idiom*$_x$ {

For each state S_1 that satisfies *Idiom*$_x$'s *Idiom_Current_Selection_List* and *Idiom_Shottype_Combinataion* {

Let S_2 be the new Idiom State for S_1;

From state index = S_1; To state index = S_2;

Timing constraint = "*curtime* == *Idiom*$_x$'s *Idiom_Waiting_Time*";

Triggering channel = empty;

Update = “curtime = 0”;

For each visual stream *i* {

If (*i* is in *Idiom*$_x$’s *Idiom_New_Selection_List*)

Update = Update + “selected[i] = true”;

Else

Update = Update + “selected[i] = false”;

}

Output one new transition < *From state index, To state index, Timing constraint, Triggering channel, Update*>;

} }

A.3.3 Adding Transitions from Idiom States to Other States

For each Idiom State S_1 {

Let S_2 be S_1’s Corresponding Output State;

For each transition T originating from S_2 {

From state index = S_1; To state index = T’s To State Index;

Timing constraint = “*curtime == Idiom*$_x$*’s Idiom_Show_Time*”;

Triggering channel = *T*’s Triggering channel;

Update = *T*’s Update;

Output one new transition <*From state index, To state index, Timing constraint, Triggering channel, Update*>;

```
        }
}
```

Example System:

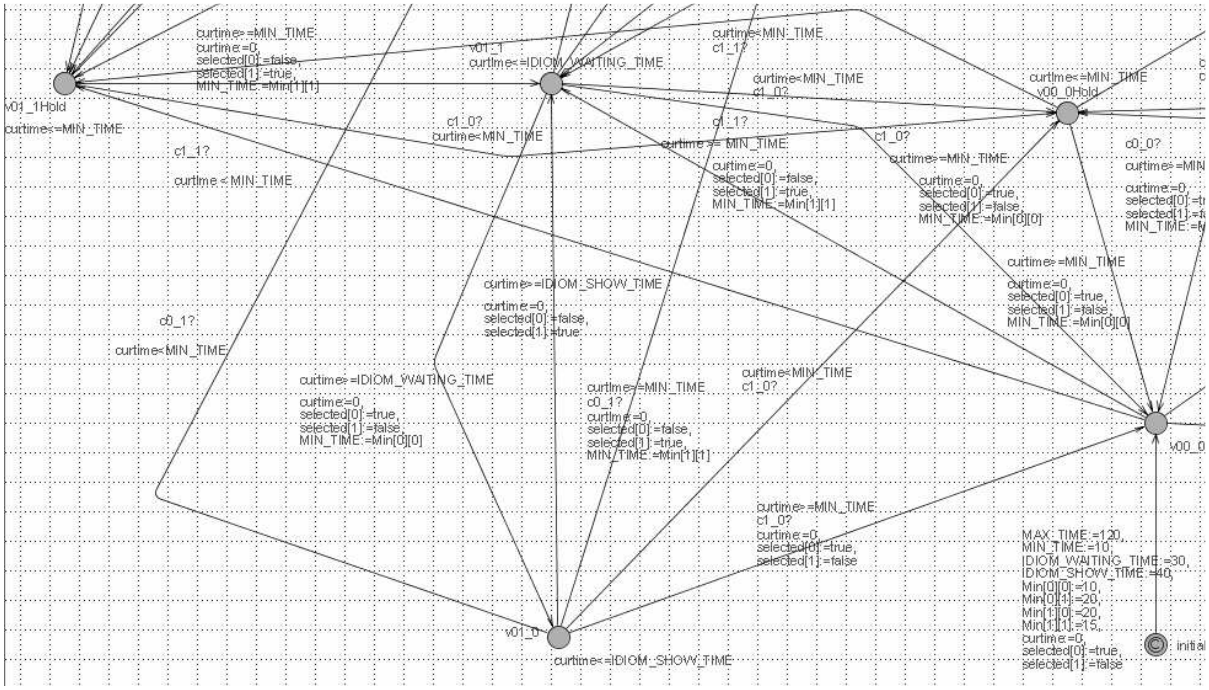

Figure 44. Illustration of Changes Added To Support Idioms

In our example, we added the support for the following idiom: if the shot type combination remains *01* (stream *0* is in shot type *0* and stream *1* is in shot type *1*) for longer than *IDIOM_WAITING_TIME*, then switch to select stream *0* for *IDIOM_SHOWING_TIME*. The changes to the automaton are illustrated in Figure 44.

The major changes are the following:

- There is a new state added *V01_0*, and there is a new transition from the state *V01_1* to *V01_0*. The idiom is enforced by the following two conditions: the state *V01_1* is

enhanced with a new invariant "*curtime* ≤ *IDIOM_WAITING_TIME*", and the transition from *V01_1* to *V01_0* has a condition "*curtime* ≤ *IDIOM_WAITING_TIME*". Their combined effect is that the transition will take place exactly after the automaton has stayed in the *V01_1* state for *IDIOM_WAITING_TIME* and not any shot type change has occurred. There is another new transition from the state *V01_0* back to the state *V01_1*, and it is guaranteed to occur after the automaton has stayed in the state *V01_0* for *IDIOM_SHOW_TIME* if no shot type change event occurs.

- There are 5 outgoing transitions from the new state *V01_0* to all the states that *V01_0* is linked to. They represent the cases that while the automaton is in the idiom specified selection state, some shot type changing event occurs to move the state of the automaton to a hold state for some other output state.

REFERENCES

[1] Access Grid, http://www.accessgrid.org

[2] Autoauditorium System, http://www.autoauditorium.com/

[3] Bayesian Network, http://en.wikipedia.org/wiki/Bayesian_network

[4] Microsoft ConferenceXP, http://www.conferencexp.net

[5] Microsoft TV IPTV Edition, http://www.microsoft.com/tv/content/Solutions/IPTV/mstv_IPTV_Overview.mspx

[6] MSN Messenger, http://messenger.msn.com/

[7] OpenCV – Open Computer Vision Library , http://sourceforge.net/projects/opencvlibrary/

[8] OpenTV, http://www.opentv.com/solutions/showcase/sports.html

[9] Polycom, http://www.polycom.com

[10] UPPAAL, http://www.uppaal.com/

[11] R. Alur and D. Dill. *A Theory of Timed Automata*. Theoretical Computer Science, 126:183–235, 1994. Preliminary version appears in Proc. 17th ICALP,1990, LNCS 443.

[12] D. Arijon. *Grammar of the Film Language*. Communication Arts Books, Hastings House, Publishers, New York, 1976.

[13] R. Baecker, *A Principled Design For Scalable Internet Visual Communications With Rich Media, Interactivity, And Structured Archives*, in Proc. of the 2003 conference of the Centre for Advanced Studies on Collaborative research, 2003

[14] J.C.M. Baeten, M.A. Reniers, *Discrete Time Process Algebra with Relative Timing*, in Fundamentals of Computation Theory, 1995

[15] J. Boyd, E. Hunter, P. Kelly, L. Tai, C. Phillips and R. Jain, *MPI-Video Infrastructure for Dynamic Environments*, IEEE International Conference on Multimedia Systems 98, June 1998

[16] P. Chang, M. Han and Y. Gong, *Extract highlights from baseball game video with Hidden Markov Models*, in International Conference on Image Processing, 2002.

[17] M. Chen, *Design of a Virtual Auditorium*, in Proceedings of 9th ACM Multimedia, September 2001

[18] R. Cutler and L. Davis, *Look Who's Talking: Speaker Detection Using Video and Audio Correlation*, in Proceedings of IEEE International Conference on Multimedia and Expo, 2000

[19] S. M. Drucker, D. Zeltzer, *Intelligent Camera Control in a Virtual Environment*, Proceedings of Graphics Interface '94

[20] H. E. Egeth and S. Yantis, *Visual Attention: Control, Representation, And Time Course*. Annual Review of Psychology, 48, 269¨C297, 1997

[21] L. Gharai, C. Perkins, R. Riley and A. Mankin, *Large Scale Video Conferencing: A Digital Amphitheater*. In Proceedings of the 8th International Conference on Distributed Multimedia Systems, September 2002.

[22] E. Hjelmas and B.K. Kow, *Face Detection: A Survey*, Computer Vision and Image Understanding, 83, pp.236-274, (2001)

[23] H.P. Kelly, A. Katkere, Y. D. Kuramura, S. Moezzi, S Chatterjee, R. Jain, *An Architecture for Multiple Perspective Interactive Video*. In Proceedings of the third ACM Multimedia, November 1995

[24] T. Ishihara and N. Otsu, *Gesture Recognition Using Auto-Regressive Coefficients of Higher-Order Local Auto-Correlation Features*, in Proceedings. Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 17-19 May 2004

[25] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale and S. Shafer, *Multi-camera Multi-person Tracking for EasyLiving*. In Proceedings of the Third IEEE International Workshop on Visual Surveillance, pages 3--10. IEEE Computer Society, 2000.

[26] S. Kawato and J. Ohya. *Real-Time Detection Of Nodding And Head-Shaking By Directly Detecting And Tracking The "Between-Eyes"*, in Proceedings of IEEE 4th International Conference on Automatic Face and Gesture Recognition, 2000.

[27] M. Li, D. Li, N. Dimitrova and I. Sethi, *Audio-Visual Talking Face Detection*, in Proceedings of IEEE International Conference on Multimedia and Expo, 2003

[28] Y. Li, T. Zhang and D. Tretter, *An Overview Of Video Abstraction Techniques*. HP Laboratory Technical Report, HPL-2001-191, 2001.

[29] Q. Liu, D. Kimber, J. Foote, L. Wilcox and J. Boreczky, *FLYSPEC: A Multi-User Video Camera System with Hybrid Human and Automatic Control*, In Proceedings of the Tenth ACM International Conference on Multimedia (MM02), France, 2002

[30] X. Liu and K. Fujimura, *Hand Gesture Recognition Using Depth Data*, in Proceedings. Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 17-19 May 2004

[31] Y. Ma, L. Lu, H.J. Zhang, M. Li, *A User Attention Model For Video Summarization*, in Proceedings of the tenth ACM international conference on Multimedia, December, 2002.

[32] A. Mojaev and A. Zell, *Real-time Face Tracking Using Discriminator Technique On Standard PC Hardware*, Intelligent Robots and Systems, 2004. (IROS 2004)

[33] T. Murata, *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE Volume 77, Issue 4, April 1989 Page(s):541-580.

[34] K. Nummiaro, E. Koller-Meier, T. Svoboda, D. Roth and L. Van Gool, *Color-Based Object Tracking in Multi-Camera Environments*, 25th Pattern Recognition Symposium, DAGM 2003, Magdeburg, Germany, September 2003

[35] S. Pfeiffer, R. Lienhart, S. Fischer and W. Effelsberg, *Abstracting Digital Movies Automatically*, in Journal of Visual Communication and Image Representation. Vol. 7, Nr. 4, S. 345-353, 1996

[36] RW Remington, JC Johnston, S. Yantis, *Involuntary Attentional Capture By Abrupt Onsets*. Percept. Psychophys. 51:279¨C90, 1992

[37] M. Roman, *An Application Framework for Active Space Applications*, PhD Thesis, 2003

[38] Y. Rui, A. Gupta, J. Grudin and L.W. He, *Automating Lecture Capture And Broadcast: Technology And Videography*, in ACM Multimedia Systems Journal (Springer), 10:3-15 2004

[39] H. Schneiderman, *Feature-Centric Evaluation for Efficient Cascaded Object Detection*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, June, 2004

[40] D. Song and K. Goldberg, *ShareCam Part I: Interface, System Architecture, and Implementation of a Collaboratively Controlled Robotic Webcam*, in Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, USA, 2003

[41] T. Syeda-Mahmood and D. Ponceleon, *Learning Video Browsing Behavior and its Application in the Generation of Video Previews*, in Proceedings of the ninth ACM international conference on Multimedia, October 2001.

[42] J. Theeuwes, *Perceptual Selectivity Is Task-Dependent: Evidence From Selective Search*. Acta Psychol. 74:81¨C99, 1990

[43] W.K. Teo, L. Silva and P. Vadakkepat, *Facial Expression Detection and Recognition System*, Journal of The Institution of Engineers, Singapore, Vol. 44 Issue 3 2004

[44] C. Toklu, S.P. Liou and M. Das, *Videoabstract: A Hybrid Approach To Generate Semantically Meaningful Video Summaries*, in Proceedings of IEEE International Conference on Multimedia and Expo, 2000.

[45] C.D. Wickens, J. Helleburg, J. Goh, X. Xu and W.J. Horrey, *Pilot Task Management: Testing An Attentional Expected Value Model Of Visual Scanning* (Tech. Report ARL-01-14/NASA-01-7). Savoy, IL: University of Illinois, Aviation Research Laboratory, 2001

[46] M-H Yang, D. J. Kriegman and N. Ahuja, *Detecting Faces in Images*, A Survey, IEEE Trans. PAMI, Vol.24, No.1, pp.34-58, Jan, 2002

[47] S. Yantis, *Goal Directed And Stimulus Driven Determinants Of Attentional Control*. In S. Monsell & J. Driver (Eds.), Attention & performance XVIII (pp. 73-104). Cambridge, MA: MIT Press, 2000

[48] B. Yu and K. Nahrstedt, *AVPUC: Automatic Video Production with User Customization*, in Twelfth Annual Multimedia Computing and Networking Conference (MMCN '05)

[49] B. Yu, W. Y. Ma, K. Nahrstedt and H. J. Zhang, *Video Summarization Based on User Log Enhanced Link Analysis*, in Proceedings of ACM Multimedia 2003, November 2003

Author's Bibliography

Bin Yu was born in Beijing, China in November 1977. He received his Bachelor of Science degree in Computer Science from Tsinghua University, Beijing, China in 2000. Then he was admitted to the Department of Computer Science in the University of Illinois at Urbana-Champaign, and joined the MONET (Multimedia Operating Systems and Networking) research group headed by Professor Klara Nahrstedt. Bin Yu has a wide range of research interests in Multimedia area, such as video coding and streaming and real-time collaboration systems.

Besides a strong academic background, Bin Yu also has extensive industry experiences during his PhD study. In summer 2002, he worked as a researcher intern at Microsoft Research Asia in Beijing, under the mentoring of Wei-Ying Ma and Hong-Jiang Zhang. In summer 2004, he worked as a researcher intern at Microsoft Research in Redmond, under the mentoring of Yong Rui. In summer 2005, he worked as a researcher intern in Microsoft Research, under the mentoring of Cha Zhang, Yong Rui and Patrick Bristow.

Bin Yu is a recipient of Carver Fellowship in the College of Engineering at UIUC in 2005, and he is awarded the title of "Emerging Leaders in Multimedia" by IBM Research in summer 2005. After graduation, he will join Morgan Stanley as a quantitative analyst.